



Secure Communication in Disaster Scenarios

Dissertation

zur Erlangung des Doktorgrades der Naturwissenschaften
(Dr. rer. nat.)

dem Fachbereich Mathematik und Informatik
der Philipps-Universität Marburg
vorgelegt von

Diplom-Informatiker

Lars Baumgärtner

geboren in Offenbach

Marburg, im September 2018

Vom Fachbereich Mathematik und Informatik der Philipps-Universität Marburg
(Hochschulkennziffer 1180) als Dissertation am 3. September 2018 angenommen.

1. **Gutachter:** Prof. Dr. Bernd Freisleben, Philipps-Universität Marburg
2. **Gutachter:** Prof. Dr. Matthias Hollick, Technische Universität Darmstadt

Tag der Einreichung am 3. September 2018.
Tag der mündlichen Prüfung am 27. November 2018.

Eidesstattliche Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig, ohne unerlaubte Hilfe angefertigt und mich dabei keiner anderen als der von mir ausdrücklich bezeichneten Quellen und Hilfen bedient habe. Die Dissertation wurde in der jetzigen oder einer ähnlichen Form noch bei keiner anderen Hochschule eingereicht und hat noch keinen sonstigen Prüfungszwecken gedient.

Marburg, den

Datum

Unterschrift

Abstract

During disasters, existing telecommunication infrastructures are often congested or even destroyed. In these situations, mobile devices can be interconnected using wireless ad hoc and disruption-tolerant networking to establish a backup emergency communication system for civilians and emergency services. When available, a connection to cloud services in the Internet is a valuable aid in crisis and disaster management. However, such communication systems entail serious security risks, since adversaries may attempt to steal confidential data, fake notifications of emergency services, or perform denial-of-service (DoS) attacks. This thesis proposes novel emergency communication approaches for challenged networks of mobile devices, addressing issues ranging from mobile device communication to cloud services running on servers in the Internet. Using these approaches, the security of mobile device-to-device communication, the security of emergency apps running on mobile devices, and the security of server systems hosting cloud services are improved.

Deutsche Zusammenfassung

Während Naturkatastrophen oder terroristischer Anschläge ist die bestehende Kommunikationsinfrastruktur häufig überlastet oder fällt komplett aus. In diesen Situationen können mobile Geräte mithilfe von drahtloser ad-hoc- und unterbrechungstoleranter Vernetzung miteinander verbunden werden, um ein Notfall-Kommunikationssystem für Zivilisten und Rettungsdienste einzurichten. Falls verfügbar, kann eine Verbindung zu Cloud-Diensten im Internet eine wertvolle Hilfe im Krisen- und Katastrophenmanagement sein. Solche Kommunikationssysteme bergen jedoch ernsthafte Sicherheitsrisiken, da Angreifer versuchen könnten, vertrauliche Daten zu stehlen, gefälschte Benachrichtigungen von Notfalldiensten einzuspeisen oder Denial-of-Service (DoS) Angriffe durchzuführen. Diese Dissertation schlägt neue Ansätze zur Kommunikation in Notfallnetzen von mobilen Geräten vor, die von der Kommunikation zwischen Mobilfunkgeräten bis zu Cloud-Diensten auf Servern im Internet reichen. Durch die Nutzung dieser Ansätze werden die Sicherheit der Geräte-zu-Geräte-Kommunikation, die Sicherheit von Notfall-Apps auf mobilen Geräten und die Sicherheit von Server-Systemen für Cloud-Dienste verbessert.

Acknowledgments

First of all, I would like to thank Prof. Dr. Bernd Freisleben for supervising me over the course of my dissertation, for his assistance, and the valuable discussions that helped me to advance this thesis. I am grateful that he gave me the opportunity to work in different projects and let me follow different, exciting research topics.

I would also like to thank Prof. Dr. Matthias Hollick at the Technische Universität Darmstadt for kindly taking the time to act as a reviewer of my thesis. Moreover, I am thankful for the opportunity to benefit from the collaboration with colleagues and researchers in the NICER project which would not have happened without him. This project was very inspirational by not only providing technically challenging research questions, but also giving me a new form of motivation by having a humanitarian goal and working on something that is supposed to save and impact lives.

During the work on this thesis, I was financially supported by the German Ministry of Research and Education (BMBF) in the ACCEPT project, the Hessische Landesoffensive zur Entwicklung wissenschaftlich-ökonomischer Exzellenz (LOEWE) in the LOEWE-Zentrum SYNMIKRO and in the LOEWE-Schwerpunkt NICER, as well as the German Research Foundation (DFG) in the SFB 1053 (MAKI).

Furthermore, I would like to thank my past and present colleagues and students at the Distributed Systems Group in Marburg who were important for many research projects I was involved in (alphabetically): Prof. Dr. Ralph Ewerth, Prof. Dr. Sascha Fahl, Dr. Niels Fallenbeck, Pablo Graubner, Christina Heitzer, Jonas Höchst, Dr. Ernst Juhnke, Nikolaus Korfhage, Patrick Lampe, Matthias Leinweber, Prof. Dr. Dorian Minarolli, Prof. Dr. Afef Mdhaffar, Dr. Markus Mühling, Alvar Penning, Falk Schellenberg, Dr. Matthias Schmidt, Nils Schmidt, Dr. Roland Schwarzkopf, Prof. Dr. Matthew Smith, Markus Sommer, Artur Sterz, and Christian Strack. Special thanks go to Mechthild Kessler for managing almost everything in administration, so that one can completely focus on research. Without her, the Distributed Systems Group would be pretty thwarted.

While being involved in different projects, I had interesting, inspirational and fun collaborations in Marburg and across universities with (alphabetically): Lars Almon, Dr. Thomas Fober, Prof. Dr. Kurt Geihs, Dr. Bastian Hoßbach, Prof. Dr. Stefan Katzenbeisser, Prof. Dr. Anja Klein, Dr. Stefan Kohlbrecher, Florian Kohnhäuser, Christian Meurisch, Prof. Dr. Mira Mezini, Ragnar Mogk, Prof. Dr. Max Mühlhäuser, Dr. Björn Richerzhagen, Prof. Dr. Guido Salvaneschi, Prof. Dr. Bernhard Seeger, Marc Seidemann, Prof. Dr. Ralf Steinmetz, Prof. Dr. Paul Gardner-Stephen, Prof. Dr. Oskar von Stryk, Milan Stute. For these, I am very grateful.

Finally, I want to thank my family for always being there for me. My parents Angelika and Wolfgang Baumgärtner supported me in all my activities from early childhood on and encouraged me to pursue all my goals in life. They made me the curious, open-minded person I am today, for which I am deeply grateful. Last but not least, I also have to thank Eva, who always has my back and helped me through many difficult times, pushing me when I was unmotivated and providing a cozy place when needed.

My Contributions

As already indicated in my acknowledgments, I am grateful to several persons who cooperated with me or who influenced my research in one way or another. Security, systems, and network research are often joint work. Therefore, pinning achievements to a single individual is not always possible, since most developments are ongoing processes, involving contributions from different participants. Furthermore, students play a vital role in implementing ideas or assisting with experimental evaluations. Since this thesis contains content from original publications, often in verbatim form, it also includes joint and sometimes practically indivisible contributions from colleagues. Therefore, I try to highlight my specific contributions as good as possible below.

Chapter 3 presents solely my views and ideas for the research topics addressed in this thesis.

Chapter 4 includes several works that are joint efforts of different researchers in the LOEWE NICER project. Although many ideas and concepts are genuinely my work, several bachelor and master students were involved in the experimental evaluations and parts of the implementations [1]–[6]. This holds especially for Sections 4.3, 4.4, and 4.6 where student assistants contributed to realizing and evaluating my concepts. Sections 4.2, 4.5, and 4.7 are also based on students' master theses that I supervised, where I also made contributions to the design, implementation, and evaluation in the published version of these works. I developed the idea and the concept of the publication on environmental monitoring [6] plus major parts of the implementation such as the rf95modem firmware and the BLE LoRa integration. The Serval evaluation study [2] was my idea, and I also supervised the involved students. Pablo Graubner performed the energy related evaluation in the announcement interval paper [3], while I developed the underlying software (mesher) and the announcement algorithms, and a student was involved in some implementation parts and the evaluation setup. Bringing different aspects of emergency communication together in a joint publication [7] involved people from different backgrounds. My work focused on the overall architecture, as well as the integration and evaluation of the DTN component for the mobile cloud in Section 4.8, and I also had the lead for producing the paper. Here, I developed different components of the mobile cloud, such as Serval shell integration, ServalDesktopApp, sdnatui, serval-socks-proxy etc. and evaluation helpers like core-automator. Through several joint discussions, novel concepts for secure mobile device communication were formulated between Florian Kohnhäuser, Milan Stute, Lars Almon, and me. This led to the SEDCOS publication [8] in Section 4.9 where my contributions were mainly providing a system model and input for the overall concept, plus integrating knowledge gained from my previous work on various DTN systems, which was used in initial versions of the paper. For several publications [1]–[3], [5], [6], I wrote the initial texts, while I wrote major parts of the initial texts for other publications [4], [7].

Chapter 5 focuses on joint work with other members of our research group in Marburg or with former members at other universities. In the Eve and Mallory paper

[9] presented in Section 5.2, my main contributions were performing manual app audits, writing automated tests, and developing proof-of-concept exploits such as the Zoner-AV hack featured in the paper. AndroLyze and Dynalize [10], [11] were joint efforts between Pablo Graubner, students, and me. While my focus was mostly on the static app analysis and distributed system design of AndroLyze (Section 5.3) plus assisting with requirements and developing use cases for Dynalize (Section 5.4), Pablo Graubner was responsible for the architecture and cloud integration of Dynalize. AndroLyze was also part of a bachelor thesis that I supervised. I designed and conducted the emergency app audit in Section 5.5, assisted by a student.

Chapter 6 includes cooperations between different partners of the BMBF ACCEPT project. In the mail server misuse paper [12], a bachelor student was involved in the evaluation of the results, and his bachelor thesis was also about the topic presented in Section 6.2. In the malware detection paper [13], I contributed the kernel introspection and hooking parts described in Section 6.3. In the ACCEPT system [14], [15], I mainly take credit for being involved in the overall architectural design and the design and implementation of sensors on various layers (where my colleagues Christian Strack and Matthias Leinweber also made contributions), as well as in designing the use case and developing the proof-of-concept implementation.

Contents

Abstract	vii
Deutsche Zusammenfassung	ix
Acknowledgments	xi
My Contributions	xiii
1 Introduction	1
1.1 Motivation	1
1.2 Problem Statement	4
1.3 Contributions	5
1.4 Publications	6
1.5 Outline	8
2 Background	9
2.1 Networking	9
2.1.1 Mesh Networking	9
2.1.2 Delay-/Disruption-Tolerant Networking	10
2.1.3 Serval	11
2.2 Mobile Devices	12
2.3 Emergency Communication	14
2.3.1 Federal Public Warning Systems	14
2.3.2 Infrastructureless Communication	17
2.4 Cloud Computing	19
2.5 Security	20
2.5.1 Transport Layer Security	20
2.5.2 Man-in-the-Middle Attacks	20
3 Secure Emergency Communication	23
3.1 Emergency Communication	23
3.1.1 Internet Services	23
3.1.2 Mobile Applications	24
3.1.3 Shortcomings	25
3.2 Design of a Secure Emergency Communication System	26
3.2.1 Disruption-tolerant Device-to-Device Emergency Communication	27
3.2.2 Security Vulnerability Analysis of Mobile Apps	28
3.2.3 Secure Cloud Systems	29
4 Disruption-tolerant Device-to-Device Emergency Communication	31
4.1 Introduction	31

Contents

4.2	MiniWorld - An Emulation-based Evaluation Environment	33
4.2.1	Introduction	33
4.2.2	Related Work	34
4.2.3	<i>MiniWorld's</i> Design	35
4.2.4	Implementation	37
4.2.5	Experimental Evaluation	41
4.2.6	Conclusion	46
4.3	Serval - A Robust Communication Foundation	48
4.3.1	Introduction	48
4.3.2	Related Work	49
4.3.3	Experimental Evaluation	50
4.3.4	Conclusion	58
4.4	Optimizing Epidemic Announcements	60
4.4.1	Introduction	60
4.4.2	Related Work	61
4.4.3	Design	63
4.4.4	Implementation	65
4.4.5	Experimental Evaluation	66
4.4.6	Conclusion	73
4.5	DTN-RPC - Offloading Work in Challenged Environments	75
4.5.1	Introduction	75
4.5.2	Related Work	76
4.5.3	<i>DTN-RPC's</i> Design	77
4.5.4	Implementation	81
4.5.5	Experimental Evaluation	82
4.5.6	Conclusion	88
4.6	Environmental Monitoring Platforms for Disaster Scenarios	89
4.6.1	Introduction	89
4.6.2	Related Work	90
4.6.3	Sensor Platforms for Disaster Scenarios	90
4.6.4	Implementation	92
4.6.5	Experimental Evaluation	96
4.6.6	Conclusion	102
4.7	Applications for Disaster Response: SmartFace	103
4.7.1	Introduction	103
4.7.2	Related Work	103
4.7.3	<i>SmartFace's</i> Design	104
4.7.4	Implementation	105
4.7.5	Experimental Evaluation	106
4.7.6	Conclusion	113
4.8	Applications for Disaster Response: UV4EC	114
4.8.1	Introduction	114
4.8.2	Related Work	115
4.8.3	<i>UV4EC's</i> Design and Implementation	116
4.8.4	Experimental Evaluation	121
4.8.5	Conclusion	126

4.9	SEDCOS - Secure Disaster Communication	128
4.9.1	Introduction	128
4.9.2	Related Work	128
4.9.3	System Model	129
4.9.4	Secure Key Management	129
4.9.5	Resilient Communication	131
4.9.6	Experimental Evaluation	133
4.9.7	Conclusion	134
4.10	Summary	135
5	Security Vulnerability Analysis of Mobile Apps	137
5.1	Introduction	137
5.2	TLS Usage in Android Apps	138
5.2.1	Introduction	138
5.2.2	Background	139
5.2.3	Related Work	141
5.2.4	Evaluating Android SSL Usage	142
5.2.5	MITMA Study	146
5.2.6	Limitations of our Analysis	152
5.2.7	Trouble in Paradise	152
5.2.8	Countermeasures	154
5.2.9	Conclusion	156
5.2.10	List of Apps With Broken SSL Usage	157
5.3	AndroLyze: Static Mobile App Analysis	159
5.3.1	Introduction	159
5.3.2	Related Work	159
5.3.3	<i>AndroLyze's</i> Design	161
5.3.4	Implementation	165
5.3.5	Experimental Evaluation	167
5.3.6	Conclusion	173
5.4	Dynalize: Dynamic Mobile App Analysis	175
5.4.1	Introduction	175
5.4.2	Related Work	175
5.4.3	<i>Dynalize's</i> Design and Implementation	177
5.4.4	Experimental Evaluation	181
5.4.5	Conclusion	185
5.5	Security Assessment of Emergency Apps	186
5.5.1	Introduction	186
5.5.2	Popular Emergency Apps	186
5.5.3	Common Attack Surface	187
5.5.4	Individual App Audits	188
5.5.5	Conclusion	198
5.6	Summary	200
6	Secure Cloud Systems	201
6.1	Introduction	201

Contents

6.2	Assessment of Email Delivery Security	202
6.2.1	Introduction	202
6.2.2	Related Work	203
6.2.3	An Empirical Study of SMTP over TLS	205
6.2.4	Advice for Email Providers	214
6.2.5	Conclusion	215
6.3	Hardening Server Systems	217
6.3.1	Introduction	217
6.3.2	Problem Statement	217
6.3.3	Related Work	219
6.3.4	Design	220
6.3.5	Implementation	225
6.3.6	Experimental Evaluation	228
6.3.7	Conclusion	230
6.4	Reactive Realtime Cloud Infrastructure Monitoring	231
6.4.1	Introduction	231
6.4.2	Related Work	232
6.4.3	Architecture	235
6.4.4	Example Anomaly Detection	237
6.4.5	Sensor Framework	239
6.4.6	Analysis-VM	243
6.4.7	Action Framework	251
6.4.8	Conclusion	254
6.5	Summary	255
7	Conclusion	257
7.1	Summary	257
7.2	Future Work	258
	List of Figures	259
	List of Tables	263
	Bibliography	265
	Curriculum Vitae	287

1 Introduction

1.1 Motivation

The unfortunate reality is that each year disasters and emergencies occur in many places around the world. The chart shown in Figure 1.1 displays an increasing trend for most types of natural disasters in the last decades, not even taking into account acts of terrorism or wars.

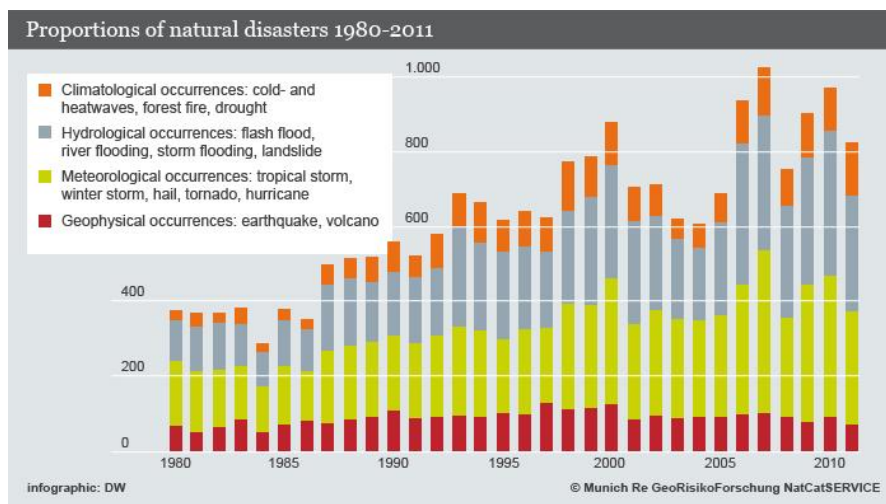


Figure 1.1: Natural catastrophes since 1980.¹

Especially highly developed countries are in danger due to their high dependency on electricity and communication. Without local food production or fuel, life depends on coordinated efforts and logistics. Apart from a lack of electrical power, another common feature of these events is that partial or complete loss of communication capacity occurs.² Even without the loss of communication capacity in a disaster, there are significant challenges to providing effective information for those affected [16]. The loss of means of communication serves to compound the difficulties and sufferings faced by those in the disaster area [17]. Furthermore, communication is needed to find missing people, self-organize aid by civilians or report local issues and emergencies to authorities. It is not only useful for person-to-person text messaging, but also vital to coordinate semi-autonomous systems such as Unmanned Aerial and Ground Vehicles (UAVs and UGVs). The use of drones and ground-based robots has many advantages and helps to protect the lives of professional responders. For example, in the Fukushima event, UGVs were sent into areas where radiation levels were too

¹Deutsche Welle: <https://p.dw.com/p/152Y2>

²<https://www.drj.com/articles/online-exclusive/when-communications-infrastructure-fails-during-a-disaster.html>

1 Introduction

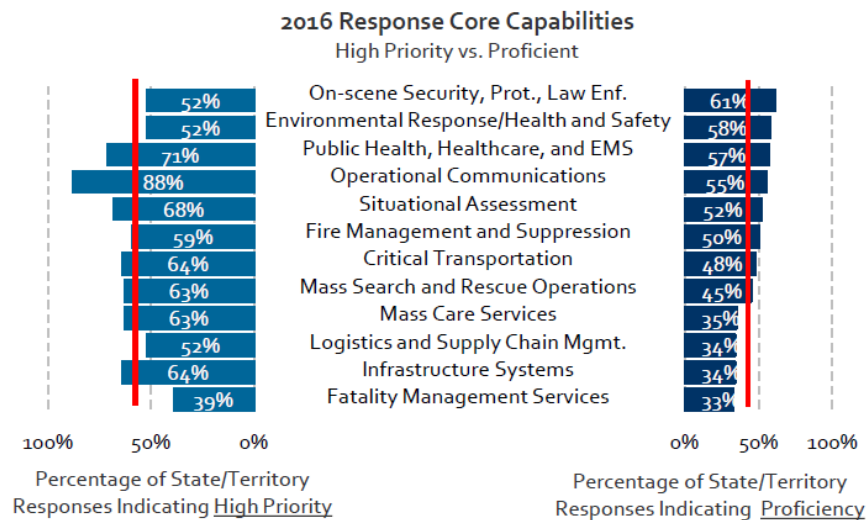


Figure 1.2: FEMA Top 10 Response Core Capabilities.⁴

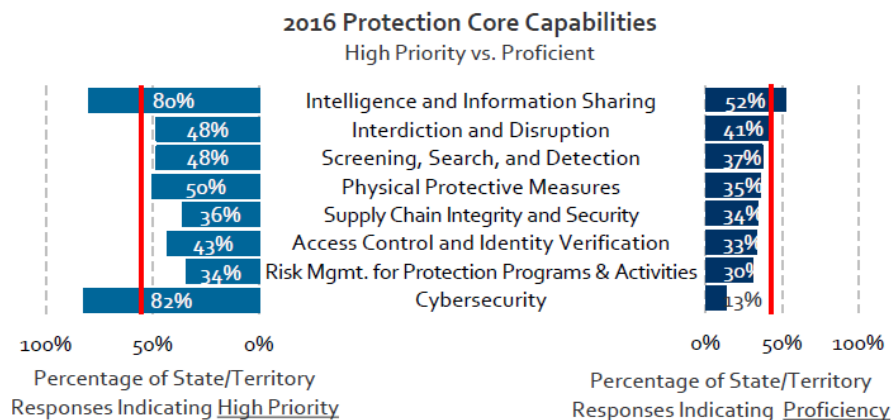


Figure 1.3: FEMA Top 10 Protection Core Capabilities.⁵

high for human rescuers, which also shows that current systems easily reach their limits.³ Furthermore, infrastructureless wireless sensing is beneficial for environmental monitoring in various application scenarios, such as in a disaster where it is vital for the operational planning of professional responders.

A communication infrastructure is a critical key component to successful disaster response. This is also reflected in the National Preparedness Report of the US' Federal Emergency Management Agency (FEMA) where operational communication has the highest priority of the 10 core capabilities (Fig. 1.2).

Since communication plays a vital role, the integrity and security of such a system

³<https://singularityhub.com/2018/04/25/how-fukushima-changed-japanese-robotics-and-woke-up-the-industry/>

⁴Source: <https://www.fema.gov/national-preparedness-report>

⁵Source: <https://www.fema.gov/national-preparedness-report>

is of importance for professional responders as well as civilians involved. FEMA's National Preparedness Report also lists cybersecurity as its top priority in its protection category (Fig. 1.3), since many other operations and tasks heavily depend on networked systems. Therefore, there is a moral imperative to seek out means of finding ways to restore, or better, sustain secure communication during and following such adverse events.

1.2 Problem Statement

To deliver a viable solution to the problems discussed above, several steps are necessary. Since the challenges for networked devices during disasters are different from regular setups, a controlled environment is needed for repeated tests, realistic development and evaluation of new approaches. Modern technology such as virtualization and emulation can be used to build simulations of larger scale with ease. This is the foundation for any further developments regarding a secure emergency communication system. Three different problem areas can be identified that are important for this thesis. First, infrastructureless communication must be used to deliver relevant services in a secure and efficient manner between end user devices. Second, the applications running on mobile devices and providing emergency services to users are critical for the security of the communication system. Third, more powerful backend systems running on machines at a mobile command center or in the cloud must be considered. Each of these areas has specific problems and requirements as outlined below.

1. **Disruption-tolerant Device-to-Device Communication.** The main challenge is to deliver services such as messaging or task offloading with remote procedure calls in a secure yet accessible way for a disaster scenario where communication infrastructure failed. Therefore, one has to rely on battery-powered commodity hardware, such as mobile phones. This limits the radio link technologies to those capable of direct device-to-device communication, such as WiFi, Bluetooth, and (with additional hardware) LoRa, without relying on working cell towers. Furthermore, the communication should ideally utilize disruption-tolerant-networking (DTN) and/or mesh networking to cope with the challenged network environment where links and devices are prone to fail or move out of range. Existing algorithms often either do not take these highly dynamic networks with often disrupted connections into account and/or neglect limited resources such as battery or computing power.
2. **Security Vulnerability Analysis of Mobile Apps.** Since emergency apps are the main interface for users on mobile devices, their security is vital for the whole system. For an in-depth analysis of the security level of any given or developed solution, dynamic as well as static code analysis should be performed to assess attack surfaces and identify vulnerabilities. Having automated tests that can be repeated easily ensures that the security level can be kept even after adding new features to an app.
3. **Secure Cloud Systems.** Having services running on larger machines in the fog/edge of a network or virtualized in the cloud also means that they are exposed to possible attacks. Protecting core services such as email, which plays a vital role for professional first responders, is important in maintaining the overall security of rescue operations. Moreover, further steps must be taken to ensure the security and integrity of such a system not only on the network level but also across the whole setup.

The aim of this thesis is to present approaches that provide solutions to the three problem areas outlined above.

1.3 Contributions

The main research contributions of this thesis are:

Topic	Contributions
<i>Emergency Network Emulation</i>	A novel, flexible, distributed emulation environment for realistically evaluating software in emergency scenarios is presented. The proposed approach utilizes full system emulation and can be integrated with external simulations for positional updates. The network back-end is flexible and can mimic different wireless interfaces.
<i>DT-D2D Communication</i>	A novel communication system including a set of new approaches to deliver a mobile cloud infrastructure with efficient data dissemination through dynamic announcement intervals and disruption-tolerant remote procedure calls is presented. These can power advanced apps such as on-device face recognition for supporting the search for missing persons, integrate UAVs for communication and rescue operations as well as facilitate various environmental sensing setups. These approaches are portable and platform agnostic, covering mobile devices, computers and small embedded system. Furthermore, a large variety of radio link technologies is covered (e.g. Bluetooth, WiFi, LoRa) by the developed solutions. Also, new solutions for increased security in the communication system are presented.
<i>Security Analysis of Apps</i>	A novel suite of tools for automated and repeatable static as well as dynamic analysis of mobile Android apps is presented. These tools can be deployed to repeatedly audit large sets of apps. The usefulness is shown through several mass audits regarding the state of security in Android apps.
<i>Secure Cloud Systems</i>	A novel approach for securing virtualized server systems by integrating classic anti-virus solutions with live system introspection and dynamic sensors deployed across several layers is presented. The use of a federated Complex Event Processing (CEP) system in conjunction with a historic event database enables new ways to eliminate false positives and detect various anomalies through minimal added logic within each virtual machine.

1.4 Publications

During the work on this thesis, the following papers were published:

1. L. Baumgärtner, A. Penning, P. Lampe, B. Richerzhagen, R. Steinmetz, and B. Freisleben, "Environmental Monitoring Using Low-Cost Hardware and Infrastructureless Wireless Communication," in *IEEE Global Humanitarian Technology Conference (GHTC 2018)*, San Jose, USA: IEEE, 2018, accepted for publication
2. R. Mogk, L. Baumgärtner, G. Salvaneschi, B. Freisleben, and M. Mezini, "Fault-tolerant Distributed Reactive Programming," in *32nd European Conference on Object-Oriented Programming (ECOOP 2018)*, vol. 109, Amsterdam, The Netherlands: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2018, 1:1–1:26
3. P. Graubner, P. Lampe, J. Höchst, L. Baumgärtner, M. Mezini, and B. Freisleben, "Opportunistic Named Functions in Disruption-tolerant Emergency Networks," in *ACM International Conference on Computing Frontiers 2018 (ACM CF'18)*, Ischia, Italy: ACM, 2018, pp. 129–137
4. L. Baumgärtner, S. Kohlbrecher, J. Euler, T. Ritter, M. Schmittner, C. Meurisch, M. Mühlhäuser, M. Hollick, O. von Stryk, and B. Freisleben, "Emergency Communication in Challenged Environments via Unmanned Ground and Aerial Vehicles," in *IEEE Global Humanitarian Technology Conference (GHTC 2017)*, San Jose, USA: IEEE, 2017, pp. 1–9
5. C. Meurisch, J. Gedeon, A. Gogel, T. A. B. Nguyen, F. Kaup, F. Kohnhäuser, L. Baumgärtner, M. Schmittner, and M. Mühlhäuser, "Temporal Coverage Analysis of Router-based Cloudlets Using Human Mobility Patterns," in *2017 IEEE Global Communications Conference: Selected Areas in Communications: Internet of Things (GlobeCom 2017 SAC IoT)*, Singapore, Singapore: IEEE, 2017, pp. 1–6
6. F. Kohnhäuser, M. Schmittner, L. Baumgärtner, L. Almon, S. Katzenbeisser, M. Hollick, and B. Freisleben, "SEDCOS: A Secure Device-to-Device Communication System for Disaster Scenarios," in *42nd Annual IEEE Conference on Local Computer Networks (LCN 2017)*, Singapore, Singapore: IEEE, 2017, pp. 195–198
7. J. Höchst, L. Baumgärtner, M. Hollick, and B. Freisleben, "Unsupervised Traffic Flow Classification Using a Neural Autoencoder," in *42nd Annual IEEE Conference on Local Computer Networks (LCN 2017)*, Singapore, Singapore: IEEE, 2017, pp. 523–526
8. A. Sterz, L. Baumgärtner, R. Mogk, M. Mezini, and B. Freisleben, "DTN-RPC: Remote Procedure Calls for Disruption-Tolerant Networking," in *IFIP Networking 2017 Conference and Workshops (Networking 2017)*, Stockholm, Sweden: IFIP, 2017, pp. 1–9
9. N. Schmidt, L. Baumgärtner, P. Lampe, K. Geihs, and B. Freisleben, "MiniWorld: Resource-aware Distributed Network Emulation via Full Virtualization," in *22nd IEEE Symposium on Computers and Communication (ISCC 2017)*, Heraklion, Greece: IEEE, 2017, pp. 818–825
10. P. Lampe, L. Baumgärtner, R. Steinmetz, and B. Freisleben, "SmartFace: Efficient Face Detection on Smartphones for Wireless On-demand Emergency Networks," in *24th International Conference on Telecommunications (ICT 2017)*, Limassol, Cyprus: IEEE, 2017, pp. 1–7
11. L. Baumgärtner, P. Graubner, J. Höchst, A. Klein, and B. Freisleben, "The More You Speak, the Less You Hear: On Dynamic Announcement Intervals in Wireless

- On-demand Networks,” in *13th Conference on Wireless On-demand Network Systems and Services (WONS 2017)*, Jackson Hole, USA: IEEE, 2017, pp. 33–40
12. L. Baumgärtner, P. Gardner-Stephen, P. Graubner, J. Lakeman, J. Höchst, P. Lampe, N. Schmidt, S. Schulz, A. Sterz, and B. Freisleben, “An Experimental Evaluation of Delay-Tolerant Networking with Serval,” in *IEEE Global Humanitarian Technology Conference (GHTC 2016)*, Seattle, USA: IEEE, 2016, pp. 1–8
 13. M. Leinweber, T. Fober, M. Strickert, L. Baumgärtner, G. Klebe, B. Freisleben, and E. Hüllermeier, “CavSimBase: A Database for Large Scale Comparison of Protein Binding Sites,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 6, pp. 1423–1434, 2016
 14. L. Baumgärtner, J. Höchst, M. Leinweber, and B. Freisleben, “How to Misuse SMTP over TLS: A Study of the (In) Security of Email Server Communication,” in *2015 IEEE Trustcom/BigDataSE/ISPA*, Helsinki, Finland: IEEE, 2015, pp. 287–294
 15. L. Baumgärtner, C. Strack, B. Hoßbach, M. Seidemann, B. Seeger, and B. Freisleben, “Complex Event Processing for Reactive Security Monitoring in Virtualized Computer Systems,” in *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems*, Oslo, Norway: ACM, 2015, pp. 22–33
 16. L. Baumgärtner, P. Graubner, N. Schmidt, and B. Freisleben, “AndroLyze: A Distributed Framework for Efficient Android App Analysis,” in *IEEE 2nd International Conference on Mobile Services (MS 2015)*, New York City, USA: IEEE, 2015, pp. 73–80
 17. P. Graubner, L. Baumgärtner, P. Heckmann, M. Müller, and B. Freisleben, “Dy-nalize: Dynamic Analysis of Mobile Apps in a Platform-as-a-Service Cloud,” in *IEEE 8th International Conference on Cloud Computing (CLOUD 2015)*, New York City, USA: IEEE, 2015, pp. 925–932
 18. M. Leinweber, L. Baumgärtner, M. Mernberger, T. Fober, E. Hüllermeier, G. Klebe, and B. Freisleben, “GPU-based Cloud Computing for Comparing the Structure of Protein Binding Sites,” in *6th IEEE International Conference on Digital Ecosystems Technologies (DEST 2012)*, Campione d’Italia, Italy: IEEE, 2012, pp. 1–6
 19. S. Fahl, M. Harbach, T. Muders, L. Baumgärtner, B. Freisleben, and M. Smith, “Why Eve and Mallory Love Android: An Analysis of Android SSL (In)Security,” in *Proceedings of the 2012 ACM Conference on Computer and Communications Security (CCS)*, Raleigh, USA: ACM, 2012, pp. 50–61
 20. L. Baumgärtner, P. Graubner, M. Leinweber, R. Schwarzkopf, M. Schmidt, B. Seeger, and B. Freisleben, “Mastering Security Anomalies in Virtualized Computing Environments via Complex Event Processing,” in *Proceedings of the The Fourth International Conference on Information, Process, and Knowledge Management (eKNOW 2012)*, Valencia, Spain: IEEE, 2012, pp. 76–81
 21. M. Schmidt, L. Baumgärtner, P. Graubner, D. Böck, and B. Freisleben, “Malware Detection and Kernel Rootkit Prevention in Cloud Computing Environments,” in *19th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP 2011)*, Ayia Napa, Cyprus: IEEE, 2011, pp. 603–610

1.5 Outline

This thesis is organized as follows:

Chapter 2 introduces topics fundamental for the research in this thesis. The topics covered include emergency communication in general, mesh and delay-tolerant networking technologies as well as cloud computing and security issues.

Chapter 3 gives a more in-depth overview of the work presented in this thesis. The architecture of the proposed system as well as challenging areas covered in the following chapters are explained.

Chapter 4 includes research results obtained to provide optimized emergency communication services on mobile devices. Apart from algorithmic improvements to data dissemination and remote procedure calls, concrete applications such as mobile face detection and its integration with unmanned vehicles are discussed.

Chapter 5 presents work to improve the security of apps running on mobile devices. Audits on current mobile apps are performed as well as general frameworks for static and dynamic analysis are presented.

Chapter 6 discusses approaches for cloud service security in emergency scenarios. The technologies covered include email as well as realtime security monitoring for cloud computing and system hardening.

Chapter 7 concludes the thesis and discusses possible areas of future work.

2 Background

In this chapter, fundamental issues relevant for the research presented in this thesis are discussed. General networking technologies such as mesh routing and delay-tolerant networking as well as relevant technologies regarding mobile devices and cloud computing are explained. Furthermore, an overview of security related topics is given.

2.1 Networking

In the context of an emergency scenario, network infrastructure is not available or unable to function reliably. Due to the highly dynamic nature of the network, static routing based on OSPF¹, RIP² or BGP³ is not suited to provide stable routes for applications to perform well. One alternative is to use *mesh networking* with algorithms that are specifically tailored to provide the kind of dynamic routing that is needed in these environments. The other alternative is to rely on *Delay-/Disruption-Tolerant-Networking* (DTN) where data dissemination happens in bundles rather than packets.

2.1.1 Mesh Networking

The most common wireless mesh routing algorithms used in communities such as Freifunk⁴, GuiFi⁵, and Commotion Wireless⁶ are described below.

OLSR

The Optimized Link State Routing protocol is formally described in RFC 3626⁷. It is a proactive link-state routing mechanism working on the IP-layer and often deployed in wireless ad-hoc networks. It uses specific Hello and Topology Control (TC) messages in conjunction with periodic broadcasts to build its routing tables. This information is used by each node to compute the next hop for all nodes. Due to its proactive nature and rather large routing tables that are disseminated, OLSR requires some bandwidth and CPU power to fully function. The main implementation runs as a userland routing daemon on various platforms such as Linux, MacOS X, FreeBSD etc.

¹<https://tools.ietf.org/html/rfc2328>

²<https://tools.ietf.org/html/rfc2453>

³<https://tools.ietf.org/html/rfc4271>

⁴<https://freifunk.net/>

⁵<https://guifi.net/>

⁶<https://www.commotionwireless.net/>

⁷<https://tools.ietf.org/html/rfc3626>

2 Background

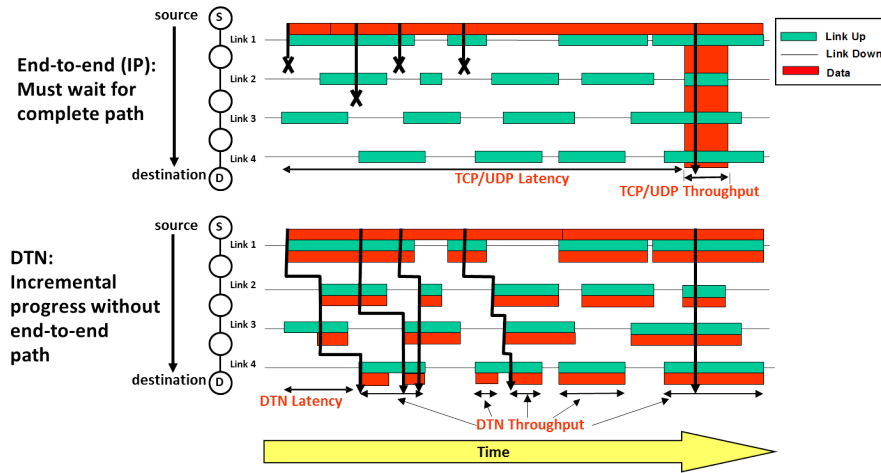


Figure 2.1: IP vs DTN data flow.¹²

B.A.T.M.A.N.

Better Approach To Mobile Adhoc Networking (B.A.T.M.A.N.)⁸ is designed as a layer 2 multi-hop routing algorithm for wireless ad-hoc networks. To eliminate the resource intensive routing information calculation and dissemination of OLSR, each node only has statistics about where it got data from and uses this for sending decisions at each hop. Therefore, the node does not know the complete best route to a destination but only the best next hop node. To build up these statistics, the network is flooded periodically with small Originator Messages (OGMs) that are forwarded and processed by the nodes. In case of multiple neighbors, a node chooses a next hop node by checking the OGM statistics for each neighbor, through which the most OGMs in the shortest time were received from the target node. The main implementation is tightly coupled to the Linux kernel and runs as a loadable kernel module in Linux.

2.1.2 Delay-/Disruption-Tolerant Networking

DTN development is strongly driven by NASA⁹ and other space agencies to ease communication between satellites, robots, spacecrafts etc. These devices only have limited communication opportunities and often unstable links. Therefore, instead of relying on real-time routing and direct connections, data is passed in a store-and-forward fashion from node to node. Here, each device acts as a "data mule", physically carrying data around. A direct comparison of end-to-end communication with IP and incremental data dissemination using DTN is shown in Figure 2.1. For reference purposes, the RFCs 4838¹⁰ and 5050¹¹ provide specifications for a *Bundle Protocol*.

There are different distribution strategies and routing algorithms specifically designed for DTNs. Some prominent replication-based candidates are summarized in the following paragraphs.

⁸<https://www.open-mesh.org>

⁹<https://www.nasa.gov/content/dtn>

¹⁰<https://tools.ietf.org/html/rfc4838>

¹¹<https://tools.ietf.org/html/rfc5050>

¹²https://www.nasa.gov/sites/default/files/dtn_0.png

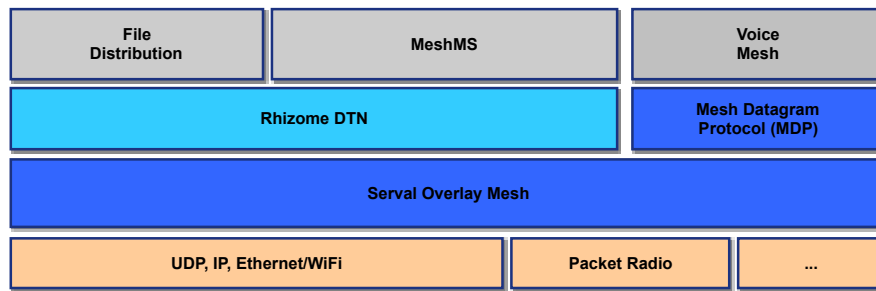


Figure 2.2: The Serval technology stack

Epidemic. Epidemic routing uses flooding as its basis, trying to synchronize all data to any new neighbor lacking a copy of it [24]. Even though many resources are wasted in this approach, it also is very robust when it comes to failing nodes and delivery rates.

PRoPHET. The Probabilistic Routing Protocol using History of Encounters and Transitivity (PRoPHET) protocol tries to conserve resources by exploiting the fact that human encounters, also in DTN “data mule” context, are rarely purely random [25]. This is achieved by keeping track of probabilities for successful delivery of bundles through different nodes. Therefore, data synchronization is only triggered if a neighbour node has a higher probability of delivering the message. This protocol has also been formalized as RFC 6693.¹³

2.1.3 Serval

Serval is centered around a suite of protocols and technologies designed to allow ad-hoc infrastructure-independent communications [26], [27], as illustrated in Fig. 2.2. The goal is to provide infrastructure-independent versions of many of the services that are commonly used on smartphones in conjunction with the Internet and/or cellular networks, e.g., voice calls, short text messaging (SMS), voice mail, social media, as well as file and image transfer.

The Serval Mesh protocols purposely take a contrasting approach to that of using IP (v4 or v6) as the basis for forming mobile ad-hoc communications networks (MANETs) [28]. The reason for this is that despite billions of dollars of research and development work, IP-based MANETs still struggle, and face a number of significant challenges that limit their real-world use, e.g., address allocation, the need to maintain a routing table, authenticity and integrity of communications, and the need for relatively reliable and stable end-to-end connectivity for such systems. Instead, Serval uses 256-bit public cryptographic keys as the primary network identifier, the so-called Serval ID (SID), and also includes a rich security model that facilitates confidentiality, integrity and authenticity by design, and does not require a Trusted Third Party (TTP) to operate. It also includes a store-and-forward DTN protocol (Rhizome), allowing network operation in the absence of end-to-end connectivity.

¹³<https://tools.ietf.org/html/rfc6693>

2 Background

Rhizome and Delay Tolerant Networking

Rhizome is a simple bundle protocol that principally defines data units as *bundles*, consisting of an optional payload, together with a *manifest* that contains necessary meta-data. *Manifests* have a hard size limit of 1 KB to improve efficiency, and must also contain a cryptographic public key that is used to protect the integrity and authenticity of the manifest itself. The manifest may also contain a cryptographic hash, indicating that it has an associated payload, together with other meta-data, such as mime-type, Rhizome service tag, file-name, and SID of the sender and/or recipient, as appropriate.

While the Rhizome implementation includes several transports for Rhizome, including HTTP, packet radio and the Serval MDP protocol described below, the protocol is purposely agnostic of the transport, to allow other transports to be added. The intention of this is that any transport that is capable of carrying bytes of data can be used to transport Rhizome data.

As a simple state-less flooding protocol, Rhizome requires no routing table, and never requires that two parties have an end-to-end connection for them to communicate. That is, the Rhizome protocol is always focused on single-hop communications, with multi-hop communications emerging as a natural consequence of *bundles* replicating among nodes.

Rhizome is used as the basis for the SMS-like Mesh Messaging Service (MeshMS) [29], and file distribution, including software updates. It is also planned to implement a twitter-like micro-blogging service using Rhizome.

MDP, MSP and Node Discovery

In addition to the Rhizome DTN protocol, Serval also includes a real-time packet-switched protocol, the Mesh Datagram Protocol (MDP) that is generally similar to UDP/IP, but uses SIDs instead of IP addresses, and includes encryption, authentication and integrity features by default. The TCP-like Mesh Streaming Protocol (MSP) is layered atop MDP to provide reliable data streaming. Various services can be implemented atop MDP and MSP, including the VoIP-like Voice over MDP Protocol (VoMP).

MDP routing uses an OSLR- and BATMAN-inspired [30], [31] ad-hoc protocol for both node discovery and maintaining a routing table, that facilitates multi-hop routing of packets. In order to reduce packet sizes, address abbreviation is used, so that only the minimum number of bytes of a SID is required to uniquely identify a node among its direct, i.e., 1-hop neighbours. This reduces the header size in the common case to be smaller than that used for IPv6.

2.2 Mobile Devices

Mobile devices in general characterize networked appliances such as tablet computers and smartphones, in a broader sense also smart watches and notebooks. In this thesis, the focus will be on smartphones and tablets. Even though many vendors exist, the majority of devices either run Apple's iOS or Google's Android operating system. Due to its openness and easy accessibility, Android is often the platform of choice for researchers. Some brief information about Google's system is given below.

AndroidManifest.xml	8 KB	XML Document
assets	--	Folder
build-data.properties	939 bytes	Visual...cument
classes.dex	6,4 MB	Document
gettingstarted.html	15 KB	HTML
jsr305_annotations	--	Folder
META-INF	--	Folder
CERT.RSA	1 KB	Document
CERT.SF	84 KB	Document
MANIFEST.MF	84 KB	mf
services	--	Folder
res	--	Folder
resources.arsc	549 KB	Document
stylesheet.css	10 KB	CSS

Figure 2.3: Contents of NINA APK

Android

Android is an operating system developed by Google specifically for smartphones, tablets and similar devices.¹⁴ At its core it uses a modified Linux kernel with a custom userland and user interface. While apps are generally written in Java, there is no stack-based Java Virtual Machine (JVM) involved, but a register-based one called Dalvik Virtual Machine (Dalvik VM). Inter-process communication between apps can happen through explicit and implicit intents. Through this system, an app can subscribe to events such a shared photo or contact information. Sending apps can either specify an explicit application to handle the payload in the intent or just share the data such that any application can respond to it. While UI code is written in the Java language, there is also a native code interface to call binary libraries developed, for example, in C/C++. To ease development and debugging, Google provides a QEMU-based Android device emulator for all major desktop operating systems.

APK - Android Application Package

Android apps are packaged as APKs, combining all necessary resources and code in one ZIP-compressed file ready for distribution. The basic layout of a typical application such as the emergency warning application NINA is shown in Figure 2.3. While executable code for the Dalvik VM can be found in *.dex* files, native libraries are organized under a *lib* folder structured by target architecture, since one APK can be deployed on different architectures (arm, mips, x64 etc). Assets and resources necessary have their predefined locations in the APK. Moreover, resources are also available in a precompiled form in *resources.arsc*. General information about an app are given in the binary XML file called *AndroidManifest.xml*. Here, version and author information as well as access rights, public intents and referenced libraries are described. The *META-INF* contains meta-information to ensure authenticity and integrity of the APK. Google requires each app to be signed by the developer prior to installation on any device.

¹⁴<https://www.android.com/>

2.3 Emergency Communication

Several systems exist for emergency communication. There are specialized systems such as TETRA¹⁵ that are only for professionals and in many places also public sirens to alert the local population. The focus of this work is more on modern approaches using commodity devices. Here, one can distinguish between mobile device-based public warning systems that get centrally orchestrated by, e.g., the government (one-way), and general communication applications for anyone with a suitable device and app installed (two-way). Both approaches are described in more detail in the following subsections.

2.3.1 Federal Public Warning Systems

Prior to omnipresent Internet access through mobile devices, governments relied mainly on the use of public broadcast services such as television and radio as well as local sirens and speakers to inform the general population. Since the time spent with classic broadcast services is declining¹⁶ and since especially younger people use the Internet, additional warning systems are needed. Due to the high smartphone penetration in the general population^{17,18} of most countries, one favored approach is to use apps with push notifications to deliver alerts. Some of the most common warning systems, mainly from a German perspective, are presented below.

DWD WarnWetter

WarnWetter (Fig. 2.4) is a weather forecasting and warning app provided by the federal weather service of Germany, Deutscher Wetterdienst (DWD). It is available for Android as well as iOS. Even though no source code is provided, the raw weather and warning data is freely available through an official API and can be downloaded, for example, as a plaintext JSON file. The main features include general weather data, rain radar, high tide, flooding and avalanche information as well as alerts for the current location and subscribed regions.

BBK NINA

The Federal Office of Civil Protection and Disaster Assistance (Bundesamt für Bevölkerungsschutz und Katastrophenhilfe - BBK) has also developed an app to spread public safety information to smartphones. NINA (Notfall-Information- und Nachrichten-App) enables a user to subscribe to various regions as well as the current position to receive push alerts (Fig. 2.5). The warnings are from three different main categories: civil protection, weather, and high tide information. The weather information is fed from the same source as the WarnWetter app. Besides the warnings, the app also includes

¹⁵http://www.etsi.org/deliver/etsi_en/300300_300399/30039202/03_02_01_60/en_30039202v030201p.pdf

¹⁶<http://www.businessinsider.de/tv-vs-internet-media-consumption-average%2Dchart-2017-6>

¹⁷<https://newzoo.com/insights/rankings/top-50-countries-by-smartphone%2Dpenetration-and-users/>

¹⁸<http://www.pewglobal.org/2016/02/22/smartphone-ownership-and-internet%2Dusage-continues-to-climb-in-emerging-economies/>

2.3 Emergency Communication

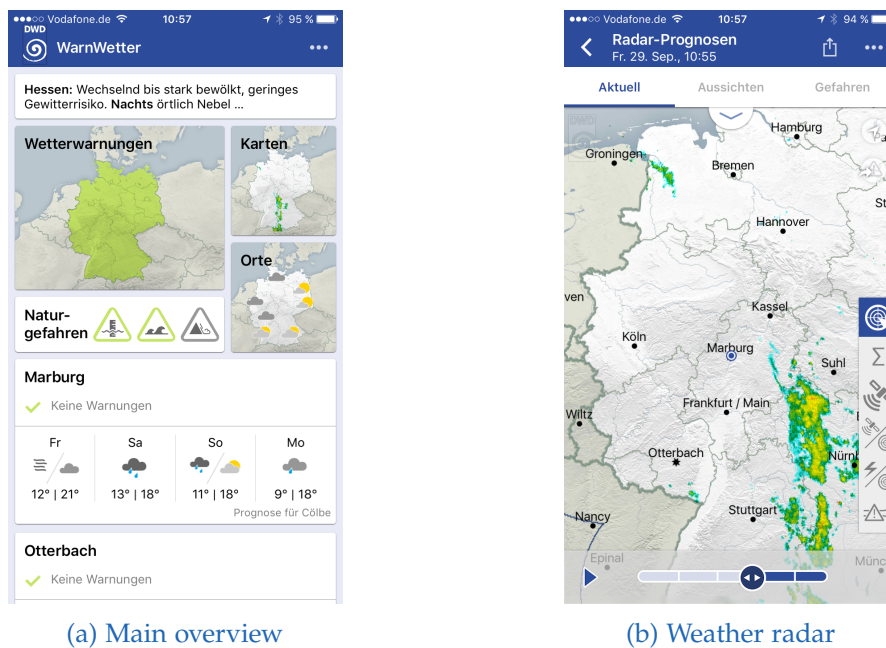


Figure 2.4: DWD WarnWetter app in action

reading material regarding various disaster scenarios and general preparedness. The app is closed source, but the alert data is freely available as JSON files to be included in other projects.

KATWARN

KATWARN (Fig. 2.6) is another system for public warnings similar to NINA. It was developed by Fraunhofer FOKUS and CombiRisk GmbH. A user can subscribe to various locations, areas or even special topics such as Oktoberfest for specific warnings. The backend also supports sending out warnings via SMS or email to users that registered for a location or topic and do not have a smartphone. KATWARN is also available internationally, for example, in Austria. Weather warnings are also issued but only for severe conditions, again sourced from the DWD. There is no source code available and also no API or direct access to the current warnings. Depending on a user's location and the corresponding districts and independent municipalities, warnings might get issued only over NINA or KATWARN or both. Therefore, both apps are needed to be safe when traveling through the whole country. Also, due the central architecture deployed, KATWARN was overloaded during the Munich terrorist attack in 2016.¹⁹

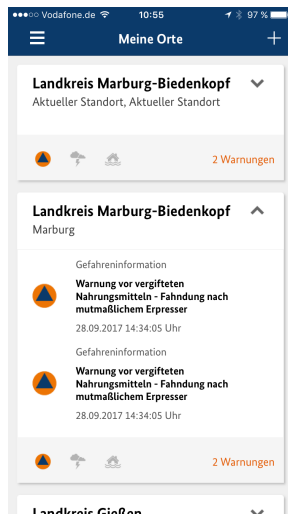
Wireless Emergency and AMBER Alerts

Wireless Emergency Alerts (WEA) / Commercial Mobile Alert System (CMAS) is a US-centric alert system designed for mobile phones utilizing the GSM infrastructure to deliver text messages via Cell Broadcast directly to cell phones.²⁰ The system was

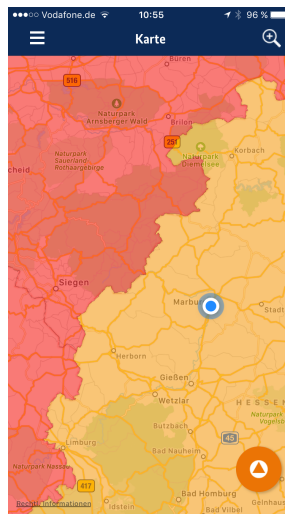
¹⁹<https://heise.de/-3277214>

²⁰https://www.fema.gov/pdf/emergency/ipaws/cmas_factsheet.pdf

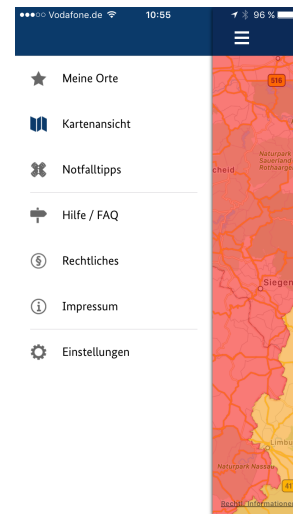
2 Background



(a) Warnings overview

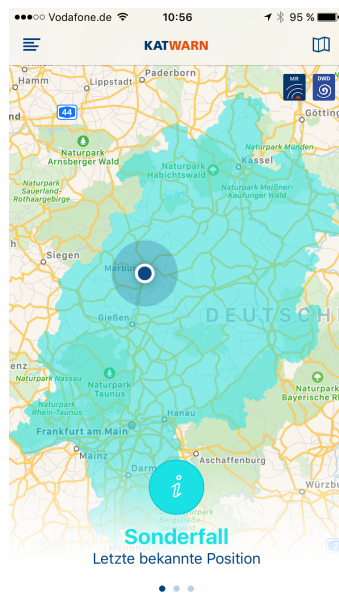


(b) Detailed map view

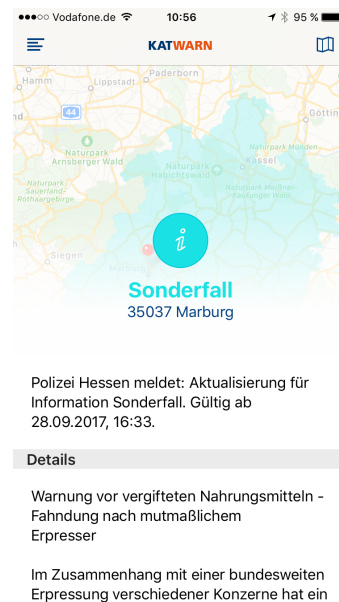


(c) Main menu

Figure 2.5: BBK NINA in action



(a) General map view



(b) Alert details

Figure 2.6: KATWARN in action

designed to be used in the following use cases:

- Alerts given out directly from the President of the United States of America.
- Warnings for imminent threats such as terrorist attacks or extreme weather.
- AMBER alerts - in case of local child abduction.

Prominent use of the system was made during the Boston marathon bombing and by the National Weather Services for various extreme weathers like tornadoes, dust storms and flash floods. For the system to be effective, the cooperation of mobile service providers as well as the cell phone manufacturers is required. Given that this is the case and the cell network is still up, this system is highly effective in reaching people, and no preparation steps such as installing a specific app is required by the general population.

Despite the limitations of the system, such as 90 character messages, no audio/video attachments etc., it still poses a valuable tool to reach people in specific areas. Unfortunately, the system is not deployed in Germany or other European countries, Japan uses a similar system to issue earthquake warnings. Also, it fails if the central infrastructure is disrupted.

2.3.2 Infrastructureless Communication

Besides communication from professional responders and government officials to the general public there is also the need for people to communicate with each other. People tend to self-organize help during disaster and therefore use social networks²¹ such as Twitter or Facebook. Often enough, Internet access is not or only partially available during disasters, which led to the development of messenger applications that communicate also directly from device-to-device. These apps are not only popular during disasters, but also for communication in oppressive regimes or during demonstration where free speech and communication might be restricted.

Briar

Briar²² is a rather new device-to-device messaging app. The focus of development was to provide a secure and robust way for activists and journalists to communicate. Besides Bluetooth and Wi-Fi communication, it can also utilize the Tor network for increased privacy and security while exchanging messages over the Internet. Besides classic messaging, group chats, forums, and blogs are supported by the app. The application and the protocol are completely open source, but the goal is only the integration of open platforms such as Android, leaving iOS users out. Therefore, Briar is not suited as a general purpose communication tool for the majority of smartphone users.

FireChat

FireChat²³ is a messaging app developed by the company Open Garden for decentralized communication without the need for Internet access. The app itself is available

²¹https://blog.twitter.com/official/en_in/a/2016/twitter-for-crisis-and-disaster-relief-in.html

²²<https://briarproject.org/>

²³<https://www.opengarden.com/firechat.html>

2 Background

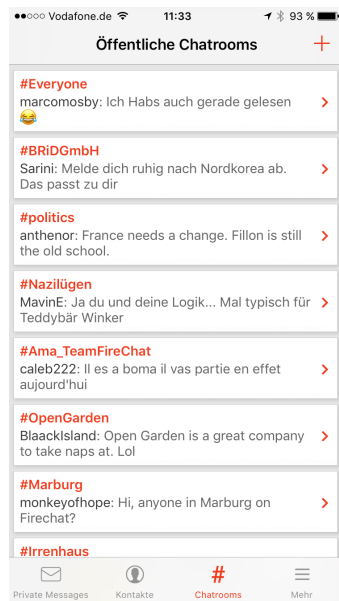


Figure 2.7: FireChat room overview

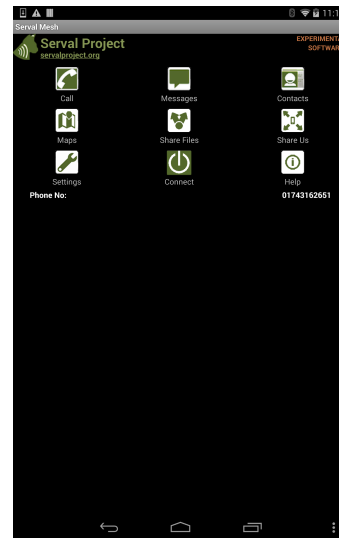


Figure 2.8: Serval Mesh main view

for Android as well as iOS platforms. It was especially popular during the Hong Kong protests²⁴ and various natural disasters. It features private one-on-one conversations as well as public chat rooms (see Fig. 2.7). The core component *MeshKit* is, as the app itself, closed source.

Serval Mesh

Serval Mesh²⁵ is an Android app (see Fig. 2.8) that uses Serval as its core to provide mesh and DTN networking through an easy app interface. It enables a phone to communicate with peers through Bluetooth, WiFi and if supported by the mobile device also ad-hoc mesh routing. The app provides mainly the following services:

- Voice calls
- Text messaging
- File sharing

The app²⁶ and the underlying system²⁷ are completely open source and development is very active even though Serval Mesh app is going to get replaced by a more messaging-focused app called Serval Chat in the future. Features such as a public twitter-like timeline missing from Serval Mesh are supposed to come with the new Serval Chat app. Strong cryptography for end-to-end encryption is already built-in and usable in Serval Mesh. The app itself has been field tested during development by the New Zealand Red Cross, among others. Various experimental ports exist, for example, for the iOS platform.

²⁴<http://edition.cnn.com/2014/10/16/tech/mobile/tomorrow-transformed-firechat/>

²⁵<https://play.google.com/store/apps/details?id=org.servalproject>

²⁶<https://github.com/servalproject/batphone>

²⁷<https://github.com/servalproject/serval-dna>

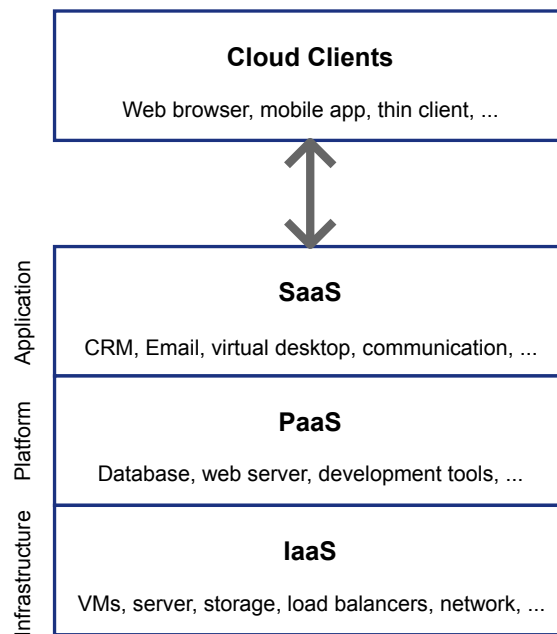


Figure 2.9: Cloud services overview

2.4 Cloud Computing

Cloud computing is a paradigm where resources are shared and provisioned according to the current requirements. Often this means not owning hardware and not having it on premise but dynamically using the resources from specialized cloud providers as a service. Through rapid provisioning and low management cost, companies can easily scale when new events like Christmas business demand other resources.

Many classic services used by companies as well as private persons, including storage, email, web-servers and databases, are migrated to cloud services with different payment models. Instead of purchasing and maintaining own hardware, only the used services in terms of bandwidth, CPU power or storage are paid. The different services (Fig. 2.9) can be categorized, e.g., as Software as a Service (SaaS), Platform as a Service (PaaS), or Infrastructure as a Service (IaaS), the latter being the closest to classic dedicated servers providing the most flexibility. A more technical look into IaaS is given in the following subsection.

Infrastructure as a Service

The purpose of IaaS is to provide a computing infrastructure on demand similar to a hosted dedicated server. This is achieved by using either hypervisor based virtualization technologies like Xen²⁸, KVM²⁹, VirtualBox³⁰, VMware ESXi³¹ or lightweight

²⁸<https://www.xenproject.org/>

²⁹<https://www.linux-kvm.org>

³⁰<https://www.virtualbox.org>

³¹<https://www.vmware.com/products/esxi-and-esx.html>

2 Background

virtualization such as Linux containers³², DragonFly vkernel³³ or FreeBSD jails³⁴. The latter shares a kernel across all guest instances, preserving resources but less flexibility and isolation for the guest user. Hypervisor-based solutions enable one to use guest OSes that are fundamentally different than the host, e.g., Windows machines running on a Linux host, and provide increased security through better isolation.

2.5 Security

2.5.1 Transport Layer Security

The Secure Sockets Layer (SSL) and its successor, Transport Layer Security (TLS), are cryptographic protocols that were introduced to protect network communication from eavesdropping and tampering. To establish a secure connection, a client must securely gain access to the public key of the server. In most client/server setups, the server obtains an X.509 certificate that contains the server's public key and is signed by a Certificate Authority (CA). When the client connects to the server, the certificate is transferred to the client. The client must then validate the certificate.³⁵ However, validation checks are not a central part of the SSL and X.509 standards. Recommendations are given, but the actual implementation is left to the application developer.

The basic validation checks include: a) does the subject (CN) of the certificate match the destination selected by the client?; b) is the signing CA a trusted CA?; c) is the signature correct?; and d) is the certificate valid in terms of its time of expiry? Additionally, revocation of a certificate and its corresponding certificate chain should be checked, but downloading Certificate Revocation Lists (CRLs) or using the Online Certificate Status Protocol (OCSP)³⁶ is often omitted. The open nature of the standard specification has several pitfalls, both on a technical and a human level. Therefore, the evaluations in the remainder of this work are based on examining the four validation checks listed above.

2.5.2 Man-in-the-Middle Attacks

In a Man-in-the-Middle attack (MITMA), the attacker is in a position to intercept messages sent between communication partners. In a passive MITMA, the attacker can only eavesdrop on the communication (attacker label: Eve), and in an active MITMA, the attacker can also tamper with the communication (attacker label: Mallory). MITMAs against mobile devices are somewhat easier to execute than against traditional desktop computers, since the use of mobile devices frequently occurs in changing and untrusted environments. Specifically, the use of open access points [32] and the evil twin attack [33] make MITMAs against mobile devices a serious threat.

TLS is fundamentally capable of preventing both Eve and Mallory from executing their attacks. However, the cases described above open up attack vectors for both Eve and Mallory. Trivially, the mixed mode/no SSL case allows Eve to eavesdrop on

³²<https://linuxcontainers.org/>

³³<https://www.dragonflybsd.org/docs/handbook/vkernel/>

³⁴<https://www.freebsd.org/doc/handbook/jails.html>

³⁵<https://tools.ietf.org/html/rfc5280>

³⁶<https://tools.ietf.org/html/rfc2560>

non-protected communication. Furthermore, Mallory might act as a transparent proxy, delivering a certificate of its own to eavesdrop on the communication.

SSL Stripping is another method by which a MITMA can be launched against an SSL connection, exploiting apps that use a mix of HTTP and HTTPS.³⁷ SSL Stripping relies on the fact that many SSL connections are established by clicking on a link in or being redirected from a non-SSL-protected site. During SSL Stripping, Mallory replaces *https://* links in the non-protected sites with insecure *http://* links. Thus, unless the user notices that the links have been tampered with, Mallory can circumvent SSL protection altogether. This attack is mainly relevant to browser apps or apps using Android's WebView.

³⁷<http://www.blackhat.com/presentations/bh-dc-09/Marlinspike/BlackHat-DC-09-Marlinspike-Defeating-SSL.pdf>

3 Secure Emergency Communication

Secure communication during emergency scenarios is as complex as it is manifold, leading to complex systems with diverse requirements. At the one end, small, resource constrained devices such as smartphones, tablets and laptops are available to civilians as well as to first responders. On the other end, large servers are relevant either in the cloud or at specialized command and control centers in the field. Since many other critical infrastructures and rescue operations heavily depend on ways of secure communication, secure emergency communication plays a vital role.

3.1 Emergency Communication

In general, there are two target groups that are relevant for emergency communication: professional responders and civilians. One issue refers to the question what happens on large, centralized systems that are connected via the Internet and provide various essential services. Another issue is what happens on devices directly available to users, most commonly mobile devices such as smartphones. Both issues have their own requirements and challenges when used for emergency communication, as described below.

3.1.1 Internet Services



Figure 3.1: Typical services used during disaster response running in the Internet.

A large variety of services exists in the Internet that are commonly used during disaster scenarios. Some of them are depicted in Figure 3.1. Several of them such as email, Google, the World Wide Web itself or social networks such as Facebook and Twitter are also heavily used in everyday life. Yet, they also play a vital role in distributing information and connecting people during an emergency, as past events

3 Secure Emergency Communication

have shown.^{1,2} They are used by professional responders as well as civilians to self-organize help. Furthermore, there are services developed and deployed specifically for disaster scenarios, e.g., to coordinate rescue teams through disaster management software such as Project Sahana's EDEN³ or managing reports in Ushahidi⁴. Other useful cloud services for such scenarios include Google's Crisis Map or Facebook's Safety Check. For rapid deployment and scalability, many of these services rely on a dynamic cloud infrastructure such as Amazon's EC2.

3.1.2 Mobile Applications



Figure 3.2: Mobile apps deployed on different devices.

A similar picture can be seen when taking a look at common mobile devices used by civilian and first responders (Fig. 3.2). Besides typical services that are also used on computers connected to the Internet, such as social networks (Twitter, Facebook etc.), the World Wide Web, or email, there are also apps more specific to smartphones, such as messengers (e.g., WhatsApp, Threema, Telegram). While the previously mentioned apps are useful in non-disaster situations as well as during a disaster, there are many apps such as Video-On-Demand (Netflix, Amazon Prime Video etc.), games and music streaming that have limited to no use during an emergency. There are a few apps specifically designed for emergency scenarios to inform citizens and to cope with the fact that less and less people can be reached by classic broadcast services such as radio and television. In Germany, examples for these apps are NINA and KATWARN that both send push notifications via the Internet to participating mobile devices. While all the previously mentioned apps on mobile devices rely on cellular networks or Internet

¹https://blog.twitter.com/official/en_in/a/2016/twitter-for-crisis-and-disaster-relief-in.html

²<https://www.fastcompany.com/40546380/facebooks-disaster-maps-helps-rescuers-know-where-theyre-needed-most>

³<https://sahanafoundation.org/>

⁴<https://www.ushahidi.com/>

access through WiFi, a few apps enable direct device-to-device communication for simple services such as text messaging (Briar, Serval Mesh, FireChat). In recent years, mobile Internet usage has surpassed that of desktop systems.⁵ Having mobile, battery powered devices readily available for the majority of people is a great advantage during an emergency scenario.

3.1.3 Shortcomings

One common requirement for many existing emergency software solutions is that they require a working communication infrastructure and a working Internet link. While most of these applications work over reliable TCP links, they also assume connections to be stable and require quite some bandwidth even for loading a simple webpage such as Facebook. When disaster strikes like, for example, hurricane Irma in 2017, this can leave even parts of a high-tech country such as the USA for a few days without connectivity. A small overview of a typical scenario can be seen in Figure 3.3.

While mobile devices are mainly used for person-to-person communication, social networks and taking footage with the built-in camera, back end services include standard communication measures such as email as well as scenario specific applications. These cloud-based services might not be accessible when communication infrastructure is destroyed during disasters, as indicated by the red arrowed connections and yellow flashes in the figure. Also, emergency warning apps such as KATWARN or NINA will not work anymore when the Internet connection is not available any longer. This is especially problematic, since they were not only meant to warn people prior to a disaster, but also to inform them during the event. Even with working connectivity, such a system might fail due to the sudden peak in user activity similar to a Denial-of-Service attack.⁶

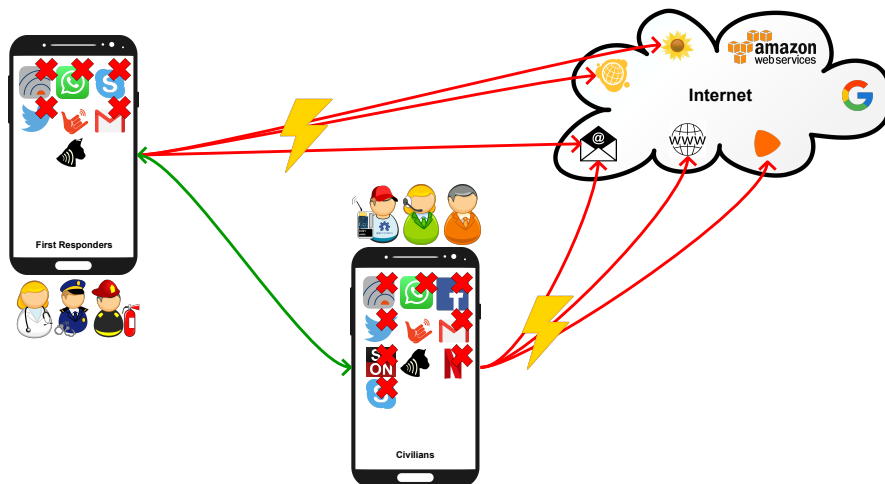


Figure 3.3: Problem overview

In the past, the public was warned and informed using, e.g., radio broadcasts, but especially young people are more focused on Internet-based media consumption and

⁵<http://bgr.com/2016/11/02/internet-usage-desktop-vs-mobile/>

⁶<https://www.focus.de/5756395>

3 Secure Emergency Communication

more and more mobile device vendors have removed the ability to use phones as FM receivers.⁷ The lack of the Internet during a disaster also renders common applications such as Skype, Twitter or WhatsApp useless. All dysfunctional apps are marked with a red 'x' in the figure. Only very few of the apps available on today's smartphones work truly offline or with local communication only. Notable examples are FireChat and Serval Mesh. If none of these are present on a smartphone, it is very limited in its usefulness during a disaster.

One of the few functions usually still working is the included camera for taking pictures or videos, but with very limited ways to share this data., e.g., via vendor specific solutions for local data exchange such as AirDrop or Google's Files Go.

3.2 Design of a Secure Emergency Communication System

The main challenge is to identify ways to re-enable communication with readily available commodity hardware during a crisis. Typically, local routers can be used together with mesh routing to form an independent communication infrastructure, e.g., Freifunk⁸ and Guifi⁹. While this approach can also be beneficial in non-disaster situations and provide stable services, it also relies on a sufficiently dense distribution of participating routers and a stable power supply. Mesh routing has the advantage that it is transparent to legacy applications, but often mobile devices lack capabilities to natively join an ad-hoc or pure 802.11s mesh network. Thus, such a network relies again on a form of maintained infrastructure.

The alternative is to use device-to-device communication, relying on the built-in direct communication capabilities of smartphones, tablets and notebooks. These technologies typically include Bluetooth, WiFi and, WiFi direct, with special application support for disruption-tolerant-networking and new mechanisms for coping with delays and disruptions. These technologies can provide a viable alternative. Furthermore, when looking at future technologies currently emerging, such as Lora, Bluetooth Mesh and 802.11ah, all mainly used for Internet-of-Things (IoT) communication, more possibilities for readily available emergency communication can be developed.

Part of this thesis' research question is to find a communication solution integrating these technologies with new approaches for secure and robust local and cloud communication.

There are three key components (see Fig. 3.4) relevant for a general emergency communication system that is suitable for civilians and professional responders alike:

- Disruption-Tolerant Device-To-Device Emergency Communication (green)
- Security Vulnerability Analysis of Mobile Apps (blue)
- Secure Cloud Systems (red)

The foundation is a disruption-tolerant device-to-device emergency communication system (indicated by green arrows in the figure) that handles various needs of users and can cope with the challenges in disaster scenarios while maintaining a high level of security. Also, the state of apps deployed on mobile devices is relevant for the overall security of the system and, therefore, must be analyzed for security vulnerabilities

⁷<http://fortune.com/2017/09/28/apple-fm-radio-hurricane/>

⁸<http://freifunk.net/>

⁹<http://guifi.net/>

3.2 Design of a Secure Emergency Communication System

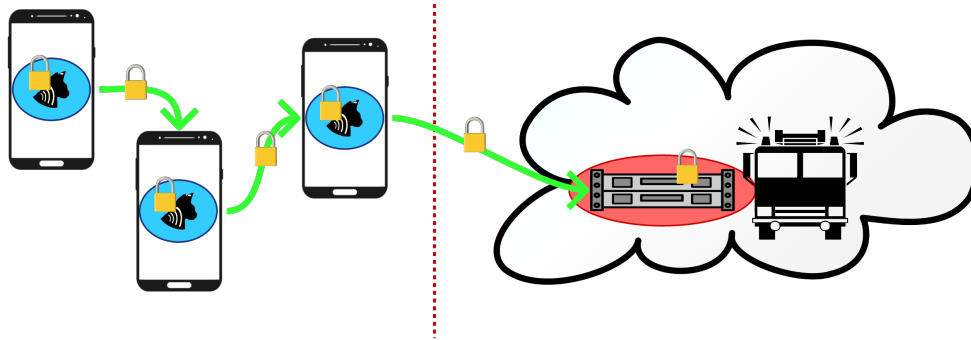


Figure 3.4: Secure communication during disaster scenarios

(marked in blue in the figure). Finally, services running on servers in the edge/fog/mist computing infrastructure (provided, e.g., by professional responders) or the cloud must be secured against failures and attackers (marked in red in the figure).

3.2.1 Disruption-tolerant Device-to-Device Emergency Communication

There are several key challenges that must be overcome to deliver a disruption-tolerant device-to-device emergency communication system that also works during a disaster scenario. These challenges are described briefly below.

Test and Evaluation Environment. Since the goal of this thesis is to deliver a system that operates in various network setups, mobility and link properties, a virtual playground for disaster scenarios is needed. While there are different network simulators available, these lack the capabilities to include software as it is and therefore, omit any side-effects caused by real world implementations of the software or the full system stack (network, kernel, userland etc.). Yet, they often provide advanced mobility patterns and (at least for WiFi) realistic communication models. On the other hand, many emulation environments are often limited in terms of mobility of the nodes or realistic simulation of link properties. To thoroughly test an emergency communication system, a simulation-emulation system with full-system emulation, flexible movement patterns, and various radio links is vital.

Basic Messaging and Data Exchange. The most basic and useful feature needed by people for organizing themselves during a crisis is some form of message exchange and a way to share binary data. Since many solutions exist that try to solve these problems, we must first evaluate those that might also work in a disaster scenario. Only peer-to-peer protocols must be considered, since we cannot rely on infrastructure during emergencies. Furthermore, not relying on TCP connections but using UDP instead helps to deal with unreliable connections. Protocols that have DTN built in are especially suitable for these setups. Further desired properties include message integrity and confidentiality that are mandatory for such a scenario.

Optimizations for Low-Bandwidth Links. Due to the fact that we heavily rely on low-bandwidth links (e.g., Bluetooth and LoRa) we have to minimize the produced traffic.

3 Secure Emergency Communication

Existing protocols often rely on flooding the network with local announcements, which works well with Ethernet or fast WiFi, but can easily overwhelm a small local network with only a few kilobits per second of bandwidth. Furthermore, some protocols such as HTTP(S) are suitable as long as the infrastructure works and bandwidth is plentiful, but for slow links HTTP(S) is very verbose, and each request and response is very large. Finally, the payload data itself is subject to optimizations. Especially large binary blobs such as image data can be optimized prior to sending either through compression or preprocessing on the device itself.

High-Level Task Offloading. Since resources such as battery power and compute power on mobile devices are considered to be very limited in a disaster scenario, alternatives must be found to handle more complex tasks. Usually, we can offload tasks by calling a remote procedure e.g., via SOAP or JSON-RPC, on a remote server and utilize its power. When we are limited to unreliable links and have highly mobile nodes, these approaches do not work. Therefore, a system is needed to deal with unknown remote-peers, disrupted connections, and delayed execution, and delayed delivery of results.

On-Device Data Processing. By processing data on a mobile device, not only the bandwidth requirements can be reduced, but also improved quality of services regarding emergency services can be delivered. One of the most common tasks during a disaster is the localization of missing persons. Therefore, having the ability to process image data on-device and detect human faces can easily be used to automate the search. Furthermore, having automated visual concept detection on-device can help to reduce the number of unimportant images that should not be spread further in the network. Since mobile devices such as smartphones or Raspberry Pis are severely limited when it comes to battery life and CPU power, algorithms must be adopted to this new environment. This also results in an overall higher processing speed, since a shorter runtime means that less power is consumed.

Secure Communication System. Besides designing the communication system to be robust against network disruptions and to work in an infrastructureless manner, it is equally important to incorporate approaches for increased security of such a system. This includes attestation of new devices during a crisis and avoiding DTN-specific attacks on message buffer management. Confidentiality and message authenticity is vital during a crisis for professionals responder as well as civilians.

Since mobile devices are key components in communication nowadays, research topics focusing on mobile communication infrastructures and application level services can be found in Chapter 4.

3.2.2 Security Vulnerability Analysis of Mobile Apps

Furthermore, the security aspects of data-at-rest and data-in-motion in emergency scenarios are vital for mobile emergency communication. Therefore, several research questions have to be answered, as outlined below.

3.2 Design of a Secure Emergency Communication System

Static Analysis of Mobile Apps. Repeatability and automation of security audits make it easy to evaluate larger sets of apps that might be relevant in an emergency scenario. While static analysis usually is a manual process on the binary, assembler or source level, the patterns of vulnerabilities found in binaries are often similar. These patterns can be related to different network usage patterns, encryption functionality, or access to sensitive information. Moreover, static analysis is suited to quickly ensure that certain mistakes are not (re-)introduced during development by reapplying the audit script to newly produced binaries.

Dynamic Analysis of Mobile Apps. There are some vulnerabilities that cannot be spotted with static analysis such as the ones nested in dynamically loaded code. For a thorough security audit, it is mandatory to also inspect apps during runtime. Since this involves on-device deployment or emulation of a mobile device, the process of automating this on a large scale is very resource intensive and quite challenging. Therefore, it is beneficial to include cloud computing resources in the design of such a system.

Security of Common Emergency Apps. There already exist various apps for communication and information spreading before, during and after disasters, as mentioned in Chapter 2. Some of them are officially distributed by governments and are widely used. By applying static and dynamic analysis together with manual audits, potential shortcomings and vulnerabilities can be identified. Having apps where one can suppress warnings or distribute false information, an attacker can potentially worsen an already critical situation or cause a disaster in the first place.

The security analysis and approaches regarding mobile devices mentioned above are presented in Chapter 5.

3.2.3 Secure Cloud Systems

The Internet provides vast resources for various services that are critical during disasters as well as everyday life. Especially due to dynamic deployments in the cloud one can easily respond to disasters and quickly get necessary services for coordination of rescue efforts running.¹⁰ Here, it is important to understand key services and find ways to protect infrastructure at risk.

Email Delivery. One of the corner stones of Internet communication is the email system based on SMTP. Its importance can also be seen as this is mentioned in the list of “basic internet communications activities like text messaging, SMS, status updates, basic web access and email” that was delivered by Google’s Project Loon to the people of Puerto Rico after being struck by a hurricane.^{11,12} By design, SMTP, responsible for the delivery of email message, is a plain text protocol. To secure a message, there are many solutions such as S/MIME or PGP/GPG that users can settle on. Factors often neglected

¹⁰<https://www.usahidi.com/plans>

¹¹<https://www.fiercewireless.com/wireless/loon-working-at-t-to-get-basic-services-to-puerto-rico>

¹²<https://blog.x.company/helping-out-in-peru-9e5a84839fd2>

3 Secure Emergency Communication

are leaking meta-data and insecure communication links. Transport-layer security is optional but critical for the overall security of emergency communication. Therefore, an evaluation of how wide-spread encrypted SMTP is, what cryptographic key size is used, who issued a certificate and what encryption ciphers are used is needed for an more in-depth understanding of the overall security of email communication.

Infrastructure Protection. All services exposed via a network are potentially at risk from malware and malicious hackers. While 100% security will never be achieved, it is vital to make a system compromise as hard as possible. If it does happen, incidents need to be detected as soon as possible to trigger counter actions. The use of virtualized systems gives a defender a variety of new options to detect attacks and infected systems, ideally, with a minimal footprint in the virtual machine and all security critical logic separated. Points to be considered are classic anti-virus protection against unknown binaries, but also advanced monitoring of network traffic, system calls or loading of kernel modules for yet unknown anomalies. By having minimally-invasive sensors across all layers of the host machine, kernel, userland and the application layer also gives us the opportunity to cross-validate findings and eliminate false positives.

New approaches regarding cloud and server security for communication systems are presented in Chapter 6.

4 Disruption-tolerant Device-to-Device Emergency Communication

4.1 Introduction

This chapter presents research conducted in this thesis regarding disruption-tolerant device-to-device emergency communication on mobile devices as well as applications and application services developed for this scenario.

First, a realistic environment for the evaluation of possible mobile communication solutions is needed. Since pure model-based simulations often differ from real world implementations and large scale physical test-beds are as expensive as they are inflexible, a solution, called *MiniWorld*, based on full system virtualization is presented in Section 4.2.

MiniWorld is used for the development and evaluation of novel DTN-based communication solutions. The Serval Project appears to be a viable basis for mobile emergency communication. To verify its claims, an in-depth evaluation regarding an emergency scenario is presented using *MiniWorld* in Section 4.3.

During this evaluation, a major issue for improvement was identified regarding Serval's energy and bandwidth requirements. So far, *Serval* uses fixed announcement intervals for peer discovery and database synchronization. Switching to dynamically adaptable announcement algorithms is much more efficient, as shown in Section 4.4.

Since the resources on mobile devices are limited, offloading tasks makes sense, but existing classical remote procedure call (RPC) solutions are not applicable in this uncertain environment. Therefore, a novel RPC system, called DTN-RPC, that is specifically tailored to DTN and D2D communication. is presented in Section 4.5.

Furthermore, with additional new components developed for embedded systems, DTN-RPC can be used for environmental monitoring in static and mobile setups, as shown in Section 4.6. By combining low-cost micro-controller units and long-range LoRa transceivers, it is possible to integrate smartphones in such infrastructureless networks. Moreover, the possibility to perform on-device visual concept detection on smartphone photos is also presented and optimized for energy efficiency on low-power devices such as the Raspberry Pi. Section 4.7 shows that on-device pre-processing of data, such as face detection, is invaluable especially for low bandwidth links.

Using the above research, it is possible to deliver new applications and network services for disaster scenarios. One common task, often heavily relying on Internet cloud services, is the detection of a missing person's face on images. A complete on-device solution for efficient face detection is presented in Section 4.7.

The real world usefulness of the developed system as a mix of mesh and DTN-based services in combination with unmanned autonomous ground and air vehicles is demonstrated in Section 4.8.

A flexible security infrastructure that is resilient to flooding attacks and that takes

4 Disruption-tolerant Device-to-Device Emergency Communication

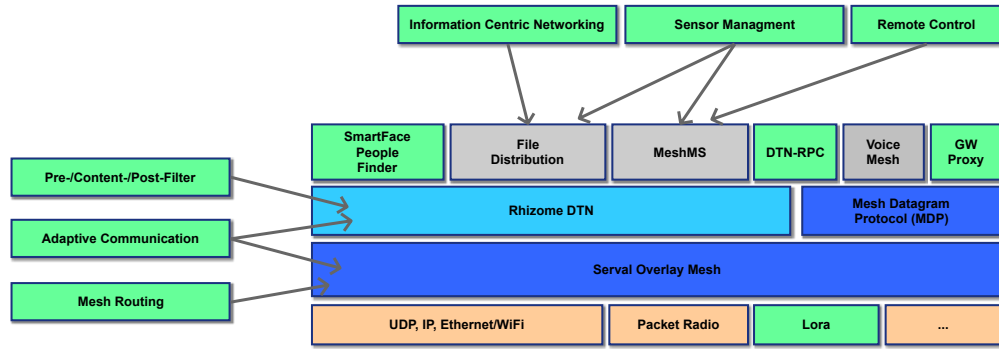


Figure 4.1: The DT D2D communication system developments highlighted in green.

proper key/certificate management into account, is necessary for any emergency communication system. *SEDCOS*, presented in Section 4.9, is a novel solution specifically designed for emergency scenarios with these problems in mind.

The various improvements and new developments in comparison to Serval are depicted in Figure 4.1. Many of the developments required for the emergency communication system described in this chapter are available as open source software. During the course of this work, contributions have been made to Serval¹ as well as CORE², through patches, bug reports, and additional software³. Regarding the *MiniWorld* emulation environment, more information can be found on github.⁴ DTN-RPC⁵ and the filters developed for integration and optimization purposes are also available.^{6,7} Various user directed developments are also available such as command line helpers and remote command execution services⁸, legacy TCP application integration⁹ as well as user interfaces.^{10,11,12} To make the most out of the various LoRa transceivers embedded on microcontrollers, a novel radio modem firmware was also developed.¹³

¹<https://github.com/servalproject/serval-dna>

²<https://github.com/coreemu/core>

³<https://github.com/gh0st42/core-automator>

⁴https://github.com/miniworld-project/miniworld_core

⁵<https://github.com/umr-ds/DTN-RPC>

⁶<https://github.com/umr-ds/serval-contentfilters>

⁷<https://github.com/umr-ds/serval-dna/tree/nicer-filters>

⁸<https://github.com/gh0st42/servalshellscripts>

⁹<https://github.com/umr-ds/python-socks5-serval>

¹⁰<https://github.com/umr-ds/serval-web>

¹¹<https://github.com/gh0st42/ServalDesktopApp>

¹²<https://github.com/gh0st42/sdnatui>

¹³<https://github.com/gh0st42/rf95modem>

4.2 MiniWorld - An Emulation-based Evaluation Environment

4.2.1 Introduction

To develop applications, algorithms, and protocols for future networks, four methods are typically used: (a) mathematical modeling, (b) network simulation, (c) network emulation, and (d) real world experiments. Mathematical modeling and network simulation are commonly used in the early stages of development, but are often criticized for inaccuracies in capturing realistic node behavior and medium characteristics [34], [35]. In contrast, real world testbeds can establish genuine environmental conditions, but are often limited in scope and induce a high management overhead.

Network emulation offers a valuable compromise between network simulation and real world experiments, since evaluations can be performed in real-time under realistic conditions and software can largely be reused when switching from emulation to real world deployment [36], [37]. In particular, in virtualization-based network emulation, real code is run in an emulated network using operating system level virtualization techniques combined with careful resource isolation and monitoring [38]–[40]. This approach provides the topology flexibility, low cost, and repeatability of simulation with the functional realism of real world testbeds.

In this section, *MiniWorld* is presented, a novel distributed network emulation framework. It is based on virtual machines (VMs) running on the cores of a shared-memory multi-core processor (centralized mode) or in a distributed system of connected multi-core processors (distributed mode). Its main properties that distinguish it from other network emulators are:

- *MiniWorld's* QEMU¹⁴/KVM-based¹⁵ full virtualization allows nearly every software and hardware to be emulated.
- Three network backends for wired and (pseudo-)wireless communication are provided to demonstrate the flexibility and modularity of *MiniWorld*.
- Four mobility patterns and three distance-based link quality models are offered to ease the development and evaluation of wireless networks.
- A *snapshot boot mode* is presented for accelerated booting of identical environments and repeating emulations.
- *MiniWorld's* emulations can be distributed across multiple computers via a resource-aware VM scheduler.
- Connection tracking, differential network switching, address configuration and network supervision features are provided automatically for every network backend.

Experimental results demonstrate the performance of *MiniWorld* with respect to VM boot times, network bandwidth, round trip times, and topology switching times, both for *MiniWorld's* centralized and distributed emulation mode.

Parts of this section have been published in [1].

¹⁴<http://www.qemu-project.org>

¹⁵<http://www.linux-kvm.org/>

4.2.2 Related Work

Several network emulators have been developed [36], [37]. For example, *CORE* [38] is a popular network emulator based on container virtualization. Since all emulated nodes share the same operating system (OS) kernel, protocol stacks other than the ones present in the kernel cannot be emulated. Further limitations are that router images, different OS and different kernels are not usable with *CORE*. In contrast, *MiniWorld* is designed for full system emulation and supports every OS and application that runs under QEMU. *CORE* offers a distributed mode in Linux based on GRE tunnels (i.e., Gretap) to connect nodes living on different emulation servers. Nodes have to be manually assigned to a specific emulation server. This is different in *MiniWorld* where a scheduler carries out the node assignment task according to resources of the emulation servers.

*Cloonix*¹⁶ is an emulator using KVM, mainly for wired networks. Several *Cloonix* servers can be interconnected, but there are fixed links between the emulated nodes, and nodes are manually pinned to their hosting server, similar to *CORE*.

DOCKEMU [39] uses *Docker* containers as its virtualization layer and utilizes Linux bridges together with the *ns-3* simulator¹⁷ to emulate PHY and MAC layers. *MiniWorld* uses the KVM instead of *Docker* containers as its virtualization layer, but adding *Docker* to *MiniWorld* would increase performance for scenarios where full system virtualization is not required.

BAMNE [41] is a network emulator that leverages a patched version of VDE [42] together with VirtualBox¹⁸. The emulator is tailored to perform tests with B.A.T.M.A.N.¹⁹. Unfortunately, no experiments that demonstrate the performance of *BAMNE* are provided.

Netkit [43] relies on full virtualization and container-based isolation. It is based on UML, a port of the Linux kernel designed to run as a user-space application. All nodes share the same file system for reading, while they use a COW mechanism to create a write-layer on a per-node basis. *Netkit* is built for educational purposes and lacks several important features, such as link emulation or a wireless mode. *MiniWorld* utilizes the COW approach in the same manner *Netkit* does.

GNS3²⁰ is an open source emulator built to help people preparing for Cisco exams. GNS3 can only emulate Cisco routers, not switches [44]. Besides *CORE* and *Cloonix*, it is the only simulator with a distributed mode.

NEMAN [45] is a network emulator for testing middleware and application layer protocols. It relies on *ns-2*²¹ scenario files for the description of mobility and topology. In contrast to *MiniWorld*, *NEMAN* requires application binaries to be modified. Furthermore, there is no virtualization layer, since processes simply bind to a tap device. Moreover, there is no link impairment and no high fidelity link emulation. *Netem*, a Linux kernel module [46], is used for link impairment by *MiniWorld*'s *Bridged LAN* and *Bridged WiFi* network backends. For high fidelity link emulation, *MiniWorld* could be combined with *ns-3*, similar to *CORE* or *DOCKEMU*.

¹⁶<http://cloonix.fr>

¹⁷<https://www.nsnam.org>

¹⁸<https://www.virtualbox.org>

¹⁹<https://www.open-mesh.org/projects/open-mesh/wiki>

²⁰<https://www.gns3.com>

²¹<http://nsnam.sourceforge.net/wiki/>

4.2 MiniWorld - An Emulation-based Evaluation Environment

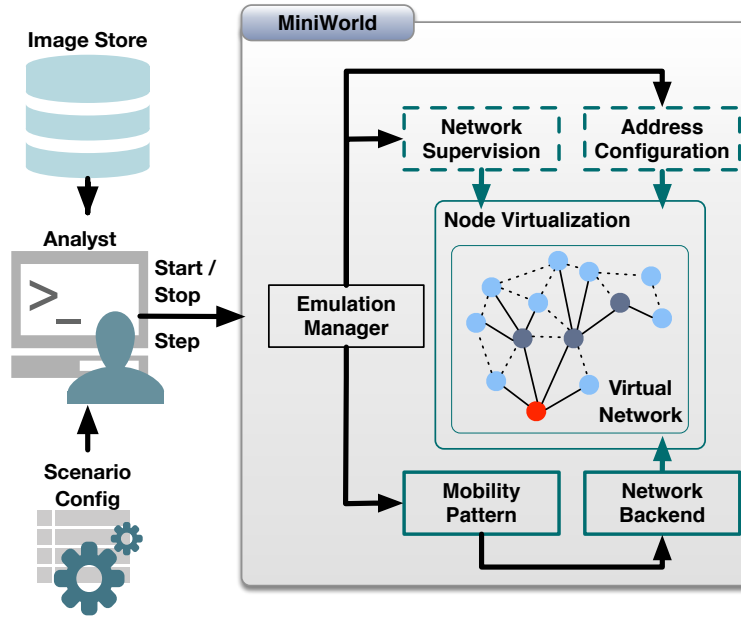


Figure 4.2: *MiniWorld's Architecture*

Mininet [40] is an emulator for SDN and comes with *open vSwitch* [47]. It uses Linux network namespaces and process isolation for each emulated node [48]. SDN controllers can run on the real network or inside the network namespaces, since *Mininet* ships with a VM with useful tools, e.g., Wireshark and dpctl, to control and view the flow tables of an OpenFlow switch. In contrast, *MiniWorld's* *Bridged LAN* and *Bridged WiFi* network backends follow the CORE approach. CORE scenario files can be built with the CORE UI and used by *MiniWorld's* CORE *Mobility Pattern* to switch between topologies.

4.2.3 MiniWorld's Design

Architecture

The architecture of *MiniWorld* is shown in Figure 4.2. To start an emulation, a network analyst needs a *Scenario Config* to setup a *Virtual Network*, and an OS image containing the software to be evaluated from the *Image Store*. The analyst starts/stops an emulation and performs emulation *steps* by invoking the *Emulation Manager*. A *step* involves the *Network Backend* to change the topology of the *Virtual Network* and requires a distance matrix from the *Mobility Pattern* as the input for the static or event-driven impairment scenario that governs node connectivity and link quality. The *Network Backend* can operate either in user-space or in kernel-space and is responsible for creating or switching the network topology according to the *Link Quality Model* and the *Mobility Pattern*. *Address Configuration* to communicate via addresses and *Network Supervision* to monitor the network topology setup are optional components.

MiniWorld needs to know when a VM has finished booting. This is accomplished by either letting *MiniWorld* know for which string it has to wait (boot mode: *Boot Prompt*) or by simulating pressing enter and waiting for the shell prompt (boot mode: *Shell Prompt*). Furthermore, network nodes are started in parallel to improve performance. After all

4 Disruption-tolerant Device-to-Device Emergency Communication

nodes have been started, the VMs are provisioned according to the shell commands supplied in the *Scenario Config*. *MiniWorld* can be set up to switch the network topology in configurable time steps (normally one second) automatically. Note that an analyst can also manually switch the network topology.

Wireless Interfaces

Virtual nodes can have multiple wireless interfaces to create separate network segments. Built-in interface types are: *AP*, *Ad-hoc*, *Mesh*, *Bluetooth* and *WiFiDirect*. Each node can have multiple instances of an interface type. An *Interface Filter* decides which interfaces are connected to each other. The default *Interface Filter* allows only interfaces of the same type and index to be interconnected. A *Network Backend* can define its own *Interface Filter* to change this behavior. A *Management* interface that is not affected by link quality impairments serves as a management/side channel that can be used in experiments for control information or SSH automation. A *Management Node* has to be provided by a *Network Backend* to support the management interface. To simulate crowded events such as football games, there is a *Hub* interface that allows a *Network Backend* to set up a single broadcast domain.

Link Quality Models

A *Link Quality Model* controls the impairment applied to the virtual network (*impairment scenario*). Static impairments serve as default values. For each step and distance between two nodes, the *Emulation Manager* calls the *Link Quality Model*. First, the model determines whether a connection will be established at all. Second, the link quality based on the distance is determined. Currently, three link quality models are available: (a) *Fixed-Range* where nodes are interconnected if their distance is less than 30 meters and the bandwidth is fixed, (b) *WiFi Simple Linear* decreases bandwidth and increases delay linearly with the distance, (c) *WiFi Simple Exponential* halves bandwidth and doubles delay every 4 meters. To prevent link quality models from slowing down the connection switching process, the link quality settings are pre-calculated (*Link Quality Caching*) for all rounded distances up to the maximum allowed connection range. This approach is a trade-off between performance and granularity.

Mobility Patterns

Currently, *MiniWorld* provides four mobility patterns: (a) *Random Walk*, (b) *Move On Big Streets*, (c) *Arma 3*, (d) *CORE Mobility*. Patterns (a) and (b) are based on OSM²² data and implement a *Random Walk* and a *Move On Big Streets* pattern. Pattern (c) is based on the *Arma 3* MilSim game²³ where coordinates are extracted from the game for each player to feed a *Movement Director* with node positions. Pattern (d) uses *CORE* scenarios exported to XML in order to have multiple topology files that can be switched after a predefined number of time steps. There are two connection modes: *LAN* and *WiFi*. The *LAN* mode considers whether nodes are connected according to the XML files. In the *WiFi* mode, the distances between nodes are used to determine link qualities between nodes. Pattern (d) can also be looped.

²²<http://www.openstreetmap.org>

²³<https://arma3.com>

Network Backends

Currently, *MiniWorld* comes with three network backends to create virtual networks, add/remove connections, and adjust link quality: (a) *VDE*, (b) *Bridged LAN*, and (c) *Bridged WiFi*.

VDE [42] is a user-space software-switch to emulate link properties, such as delay, packet loss, and duplicate packets. In *MiniWorld*'s *VDE* network backend, each interface is connected to a different *VDESwitch* to create separate network segments. A *Wirefilter* for each connection between two nodes is used to apply different link quality impairments. The *VDE* color patch²⁴ is used to create a hop-to-hop network instead of a single collision domain where all nodes can see each other. Traffic is only forwarded between switch ports if their color differs, effectively realizing a wireless mesh network.

Bridged LAN and *Bridged WiFi* leverage technologies from the Linux kernel (i.e., Linux *bridges* and Linux *TC*) to create a virtual network. *Bridged LAN* uses one interface to represent a connection. In contrast, the *Bridged WiFi* multiplexes connections via a single tap interface per *MiniWorld* interface. Thus, *Bridged LAN* is static and can be used to emulate wired networks, because the number of unique connections has to be known beforehand to set up the number of NICs in the VM. In contrast, *Bridged WiFi* is dynamic and can be used to emulate wireless networks, since it does not matter how many unique connections are going to exist.

Distributed Mode

Figure 4.3 illustrates *MiniWorld*'s distributed mode, where a central *Coordinator* allows any Linux computer (*Emulation Server*) to participate in an emulation. A user can interact with any of these via an RPC interface to query information or execute commands on specific nodes. Each *Emulation Server* operates independently for each simulation step, and virtual network nodes are hosted only on the *Emulation Servers*.

There is a score-based node placement strategy which takes the resources of the *Emulation Servers* into account. The *Coordinator* computes the distance matrix and distributes it among the *Emulation Servers*, hence the *Emulation Servers* are kept in sync with each other. Globally necessary information is shared with each *Emulation Server* before an emulation starts.

An emulation step of either the *RunLoop* or the user creates a distance matrix that is sent via ZeroMQ²⁵, a networking library for message-passing with efficient one-to-many (*Publish-Subscribe*) and one-to-one (*Request-Reply*) communication. In the *Publish-Subscribe* pattern, the distance matrix is sent to any subscriber and filtered at an *Emulation Server* for relevant data. In the *Request-Reply* pattern, the distance matrix can optionally be filtered at the *Coordinator*, since it has a separate connection to each *Emulation Server*.

4.2.4 Implementation

Figure 4.4 shows an overview of the implementation of *MiniWorld*²⁶. Its functionality, indicated by the blue buttons on the left, is accessible by an RPC interface. All classes

²⁴http://www.open-mesh.org/attachments/download/152/vde2-2.3.2_colour.patch

²⁵<http://zeromq.org>

²⁶https://github.com/miniworld-project/miniworld_core

4.2 MiniWorld - An Emulation-based Evaluation Environment

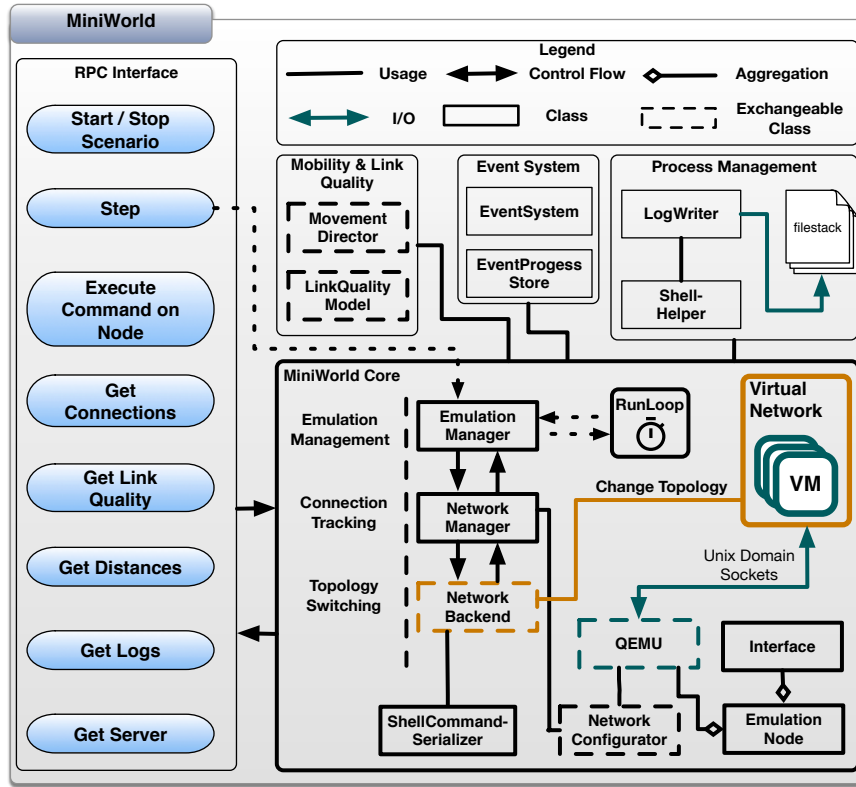


Figure 4.4: MiniWorld Implementation

ments independent of the network backend implementation. Since connections are tracked, the *EmulationManager* only communicates with the *Network Backend* if a connection changes in terms of link state or impairment (*Differential Network Switching*). An optional *Network Configurator* gives each interface of a VM an IP address. Moreover, each *Network Backend* may decide to use a custom implementation. Note that even the virtualization layer can be exchanged so that container virtualization may be added to *MiniWorld*.

Virtual Nodes

Nodes are virtualized with QEMU. *MiniWorld* does not rely on *libvirt*, instead it uses plain QEMU to leverage the full flexibility of the emulator. The QEMU command is built from options declared in the *Scenario Config*. For each different node image, an QCOW2 overlay image is generated to make use of a COW mechanism. With the COW mechanism, a single image representing a common read-only layer is used by all nodes with the same base image, hence node images are write-isolated from each other. The boot times of VMs for full virtualization are higher than for lightweight virtualization. A special boot mode called *Snapshot Boot Mode* tries to reduce these times. Snapshots are taken from the VMs. For this purpose, `{savevm,loadvm} <snapshot_name>` commands are sent to the QEMU Monitor. If an error occurs, a VM is booted normally.

4 Disruption-tolerant Device-to-Device Emergency Communication

Network Backends

VDE For each interface of the *VDE* network backend, a *VDESwitch* is started. Both the *VDESwitch* and *Wirefilter* can be controlled via an Unix Domain Socket (UDS). The *VDESwitch* interface is used to set the hub mode, manage VLAN, view switch ports and their links, set the color of each link and to set the number of ports. The *Wirefilter* UDS interface is used to define the link quality in terms of loss and bandwidth, but it supports more impairment options such as delay, duplicate packets, bandwidth, speed, noise, MTU, and more advanced modulation techniques using Markov chains.

Bridged LAN and Bridged WiFi *Bridged LAN* determines for each node the maximum number of connections during a scenario. Then, the VMs are created with the necessary number of NICs. The representation of a connection with a single NIC creates point-to-point links. Hence, for each connection, the appropriate NICs are added to a bridge. *Bridged WiFi* allows any number of connections to be multiplexed over a single NIC, hence only one NIC per *MiniWorld* interface is used. *Ebtables* is the equivalent of *iptables* to create firewalls, but operates on the link level instead. For each connection, *ebtable* rules based on the tap device names are used for filtering. Additionally, Linux traffic control facilities are used to apply different link qualities based on the connections. For each interface in *MiniWorld*, one bridge is created. Moreover, frames received by a bridge are redirected to the appropriate chain.

Virtual Network Creation and Control To create and control a virtual network, *MiniWorld* supports *Brctl*, *Iproute2* and the *Pyroute2* Python library that uses netlink sockets to interact directly with the kernel. In contrast to *Iproute2*, *Brctl* worked out of the box on all tested machines, whereas some versions of *Iproute2* do not support setting the hub mode of a bridge. Although *Brctl* does not offer a batch mode, it is included for situations where ease of use is more important than performance. Either *Brctl*, *Iproute2* or *Pyroute2* is used for creating a bridge, adding an interface to a bridge, changing the state of an interface and for transforming a bridge into a hub. *Brctl* commands can only be executed sequentially, hence communication with the kernel produces overhead for each command. *Iproute2* and *Pyroute2* are able to execute all commands at once (*batch mode*).

Link Quality Models Link quality impairment is implemented using HTB, a *classful QDisc* for bandwidth shaping. More advanced link emulation is offered by *NetEm* [46] that allows to simulate delay, packet reordering, loss and much more. Currently, only the delay is deployed in the *NetEm* capable link quality models. The delay depends to 25% on the last delay. For the *Bridged WiFi* network backend, connection flows are marked by *ebtables*. Since connections are multiplexed via a single NIC, for each connection a separate traffic class associated with the NIC is used to simulate different link quality impairments. Traffic is classified by a filter (via the flow ID) and then redirected to the appropriate traffic class. For *Bridged LAN*, a single traffic class can be used without any filtering, since each connection is represented by a NIC.

Distributed Mode

In *MiniWorld*'s distributed mode, nodes are interconnected on the link layer with GRE by default. Both sides of a connection have to establish a tunnel. The tunnel is added to a bridge on both nodes.

Scheduling To perform resource-aware placement of virtual nodes on *Emulation Servers*, their resources are transformed into a score and shared with the *Coordinator*. This includes CPU (based on *bogomips*) and RAM. The *NodePlacementScore* works as follows: nodes are placed on *Emulation Servers* based on the CPU score. RAM is only used to check that the amount of free memory is not exceeded on an *Emulation Server*. Otherwise, the number of VMs is reduced until the memory fits to the needs of the VMs. The RAM check is possible, since memory is limited for each VM.

Bridged Backends The *Bridged* network backends use *Iproute2* to set up *GreTap* tunnels such that nodes living on distinct *Emulation Servers* can be interconnected. Each tunnel requires an ID for (de)multiplexing. To avoid distributed coordination of such IDs, a pairing function is used to produce a unique ID from the two node IDs. Since *GreTap* devices are represented by a NIC on both sides of a connection, tunnels can be handled the same way as in the *centralized* mode.

Communication The central *Coordinator* distributes the distance matrix either via the *Request-Reply* or the *Publish-Subscribe* pattern among the *Emulation Servers*. After each step, *Emulation Servers* are synced via the *Request-Reply* pattern. All *Emulation Servers* subscribe to an extra channel such that a reset can be triggered at any time. The channel is implemented using the *Publish-Subscribe* pattern.

4.2.5 Experimental Evaluation

This section presents a performance evaluation of *MiniWorld* itself. In all experiments in this section, the presented values are averages of 10 repeated measurements.

Centralized Mode

The experiments in *MiniWorld*'s centralized mode are performed on a shared-memory multi-core computer. It has 16 physical CPU cores and 64 virtual cores, and 256 GiB of RAM. In the experiments, *iproute2* version 4.2.0²⁷ and QEMU version 2.6.0 are used. The used VMs are: *OpenWrt Barrier Braker* (*OpenWrtBB*; size: 68 MiB) that has been compiled manually, and *Debian 8* (*Debian8*; size: 1,568 MiB) that has been installed from a netinstall image with the options *SSH Server* and *Standard System Utilities*. The *OpenWrtBB* VM needs only 10 MiB of memory after boot, whereas the *Debian8* VM requires 78 MiB of RAM.

VM Image Boot Times The boot times of QEMU VMs in *MiniWorld* are evaluated below.

²⁷<https://kernel.googlesource.com/pub/scm/linux/kernel/git/shemminger/iproute2>

4 Disruption-tolerant Device-to-Device Emergency Communication

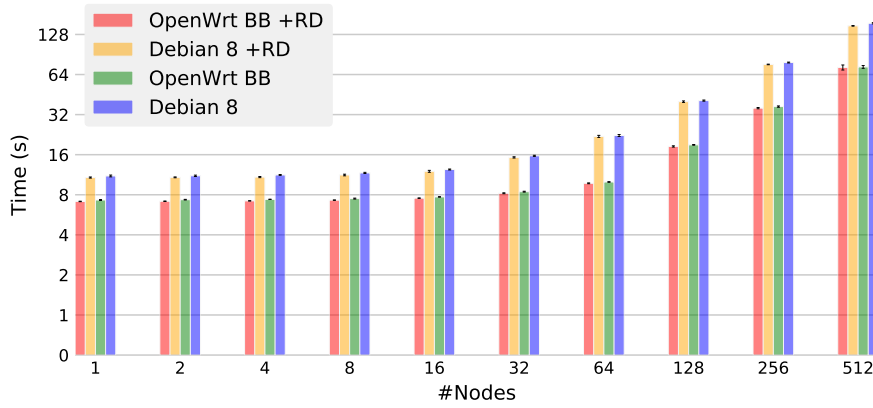


Figure 4.5: Boot Times: OpenWrtBB vs. Debian8 (*Shell Prompt*)

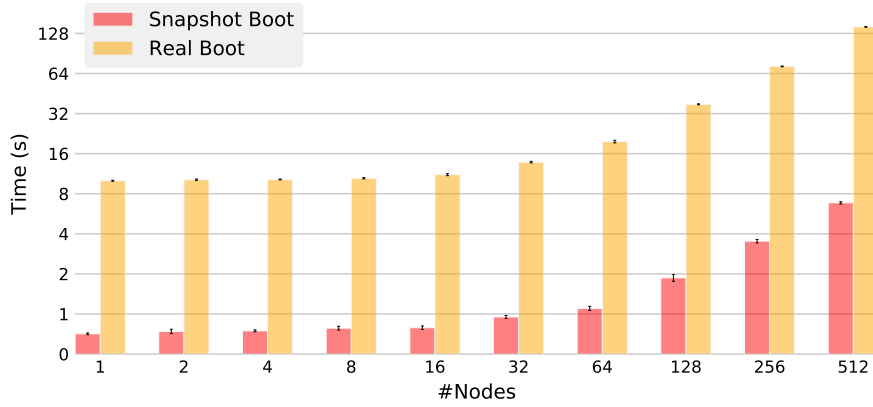


Figure 4.6: Boot Times: Snapshot Boot vs. Real Boot (Debian8)

Image Comparison Figure 4.5 shows the required boot times for both images, considering that the *Debian8* VM is equipped with 256 MiB of RAM instead of 32 MiB for *OpenWrtBB*. *OpenWrtBB* is approximately twice as fast as *Debian8*. Since the *Shell Prompt* boot mode is used, the VMs may not boot fully, but offer a shell prompt. The use of a RD reduces the boot times only slightly. Moreover, doubling the number of VMs by factor 64 (i.e., the number of virtual CPU cores) doubles the start times for each image. Therefore, *MiniWorld* scales *linearly*.

Snapshot Boot VM snapshots can be used to improve VM boot times. Snapshots are taken after the VM has been started the first time. Figure 4.6 shows the improvements of the boot times achieved by *Snapshot Boot*. Even for 512 VMs, only a few seconds are required to restore the state of the VM (6.8 seconds). A full boot requires 143.3 seconds, hence the node start times are reduced by a factor of 21.

Network Backends The three network backends are evaluated in terms of bandwidth and RTTs below. In addition, the topology switching times are measured.

4.2 MiniWorld - An Emulation-based Evaluation Environment

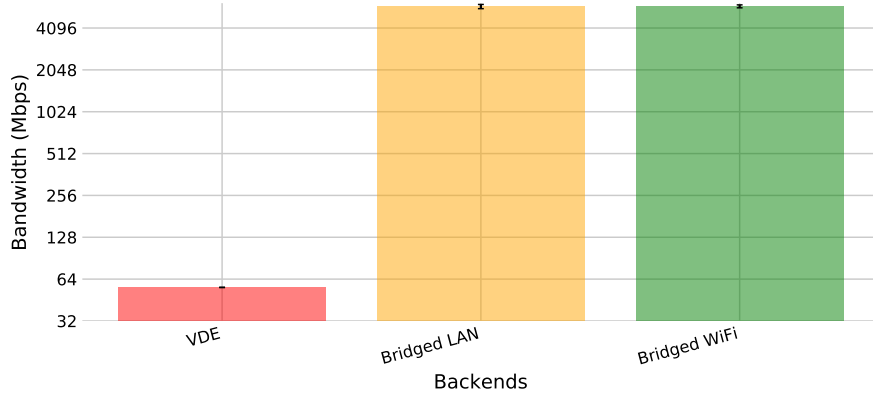


Figure 4.7: Network Backend Throughput

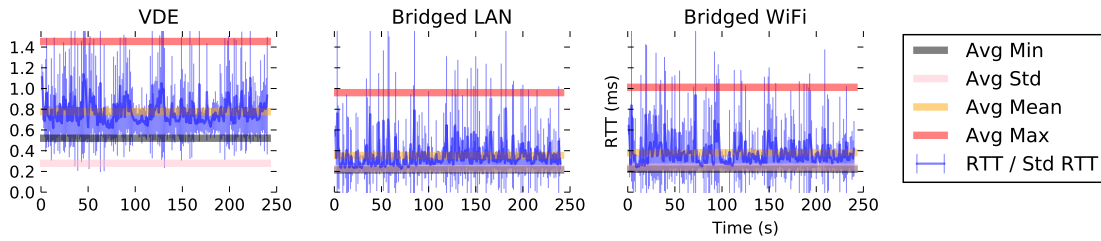


Figure 4.8: RTTs for the Network Backends

Bandwidth To measure bandwidth, two VMs and *iperf* are used. All VMs are started with 1024 MiB RAM and the *virtio-net-pci* QEMU NIC Model. Moreover, each VM gets a single CPU core. The results are shown in Figure 4.7. The *VDE* network backend provides an average bandwidth of 55.7 Mbps. The *Bridged* network backends differ only slightly from each other: *Bridged LAN* provides a bandwidth of 5848.5 Mbps, while *Bridged WiFi* offers 5867.6 Mbps of bandwidth, i.e., *ebtables* does not seem to reduce bandwidth.

Round Trip Times Figure 4.8 shows the results for the RTTs, gathered by the *ping* command over 240 seconds. All three subplots share the same *x* and *y* axis. The red line shows the average maximum delays (*Avg Max*) for 10 trials. With slightly more than 1.4 ms, the *Avg Max* (red line) is the highest for *VDE*. The *Bridged* network backends provide the same delay characteristics, since both use Linux bridges. The *Avg Mean* (yellow line) of both is less than 0.4 ms. The highest *Avg Mean* value results from the *VDE* network backend (approximately 0.8 ms), most likely since the number of ports is quite high for the *VDESwitches* by default (65537). To summarize, all network backends provide good RTTs values.

Topology Switching Mobile nodes change their positions frequently, hence switching between different topologies needs to be fast. The following experiment investigates topology switching of the *Bridged* network backends, because *VDE* did not show satisfactory results and thus cannot be used for wireless network emulation. Figure 4.9 shows the results of switching between 4 topologies, each consisting of 128 nodes:

4 Disruption-tolerant Device-to-Device Emergency Communication

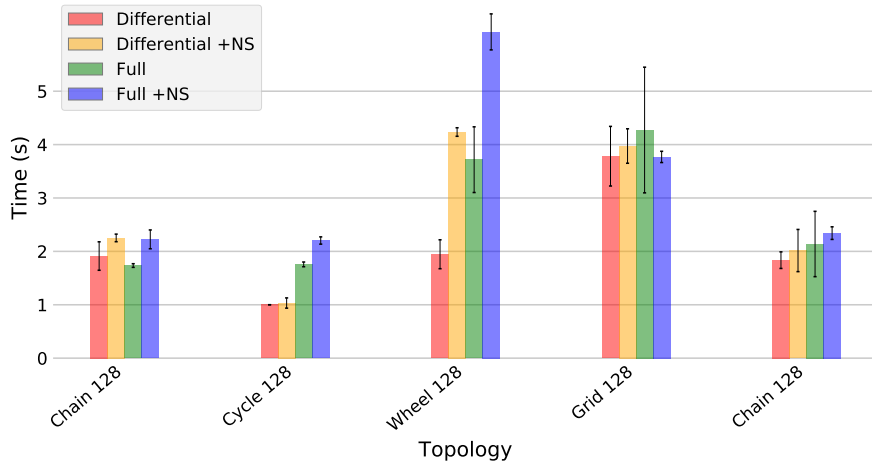


Figure 4.9: Topology Switching (Bridged WiFi)

Chain 128, *Cycle 128*, *Wheel 128*, and *Grid 128*. The first two bars of each topology show the *Differential Topology Switching* capability, since only the differences between topologies are changed. For example, switching between *Chain 128* and *Cycle 128* requires less than a second (red bar at *Cycle 128*) since the *Step Time* of the *RunLoop* is 1 second by default. Therefore, each step takes at least one second. The *Differential Topology Switching* feature is provided by the *Network Manager* and hence every network backend benefits from it automatically. The values of the green and blue lines have been created by defining the appropriate topology as the only one in the *Scenario Config*. Therefore, no *Differential Topology Switching* can be performed. In all topology switching cases depicted in Figure 4.9, *Differential Topology Switching* is faster than *Full Topology Switching*. The used network backend is the *Bridged WiFi* network backend. Moreover, Figure 4.9 illustrates that *Network Supervision* increases switching times since NICs need to be configured in terms of IP addresses, and the network connectivity needs to be checked with the *ping* command. Since *Network Checking (NC)* is an important feature to ensure that a network topology has been switched correctly, the additional times can be neglected.

Distributed Mode

In the following experiments, the distributed mode of *MiniWorld* is evaluated using 6 *Emulation Servers* and one *Coordinator*. Each of the 6 computers has a Core i7 processor with 4 physical and 8 virtual cores. Hence, 8 QEMU processes are started in parallel. Moreover, it has 32 GiB of memory and a Gigabit Ethernet card.

VM Boot Times To demonstrate *MiniWorld's* resource-aware virtual node placement on the 6 *Emulation Servers*, 300 *OpenWrtBB* nodes are started by n servers where n is increased by 1 until all 6 servers take part in the distributed emulation. The boot times (*Selectors Boot Prompt*) are shown in Figure 4.10. The start of 300 virtual nodes with a single *Emulation Server* takes 435.8 seconds. Doubling the number of servers reduces the boot times to 218.9 seconds (factor 2). With a total of 6 servers, the boot times could be lowered to 81.5 seconds. Hence, the total boot times could be reduced by a factor of 5.3.

4.2 MiniWorld - An Emulation-based Evaluation Environment

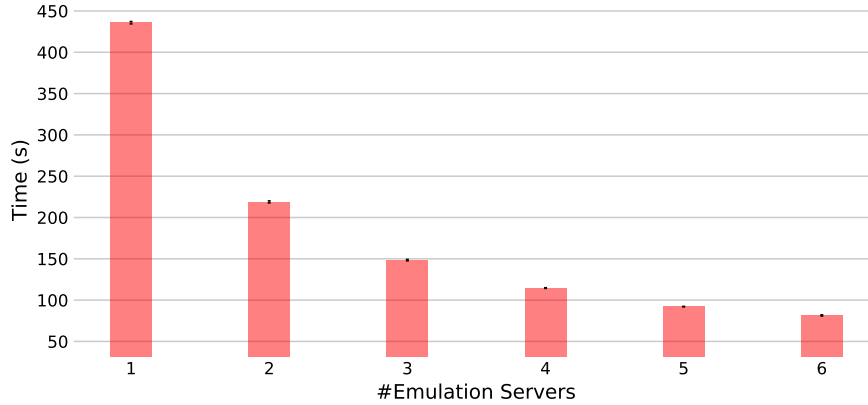


Figure 4.10: Distributed Mode: Boot Times (300 OpenWrtBB Nodes, 128 MB RAM, Selectors Boot Prompt)

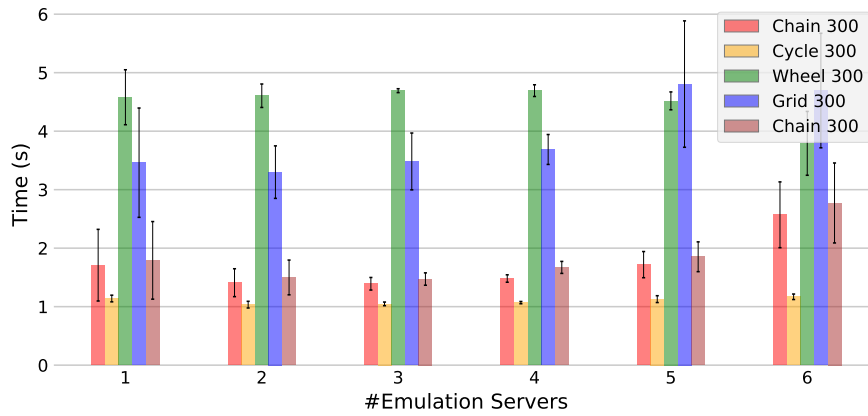


Figure 4.11: Distributed Mode: Differential Topology Switching (300 OpenWrtBB Nodes, Fixed-Range Model, Bridged WiFi Network Backend, No Link Quality Impairment)

Topology Switching The topology switching times of the distributed mode are shown in Figure 4.11. 6 servers are used to run 300 *OpenWrtBB* nodes with the *Bridged WiFi* network backend and the *Fixed-Range* model without any link impairments. Creating the *Chain 300* topology is very fast, taking less than 2 seconds. Adding only a single additional link (*Cycle 300*) can be handled efficiently by any number of *Emulation Servers*. Switching to *Wheel 300* takes the longest time. Figure 4.11 indicates that there is nearly no communication overhead. Therefore, the topology switching times in the *distributed* mode are approximately as fast in the *centralized* mode, since the number of connections to switch stay the same. One disadvantage of the *distributed* mode is that the slowest *Emulation Server* dictates the time required for a single step, since *Emulation Servers* are synced after each step.

Tunnel Delays Tunnels introduce overhead, since frames need to be encapsulated by a PDU. For *Gretap*, each frame is wrapped into an IP packet. The following experiment examines the overhead posed by tunnels on bandwidth and delay. A *Chain 6* topology is used, delay is measured with the *ping* command, and bandwidth is measured with *iperf*. A connection from node 1 to node 6 is established and data is recorded during

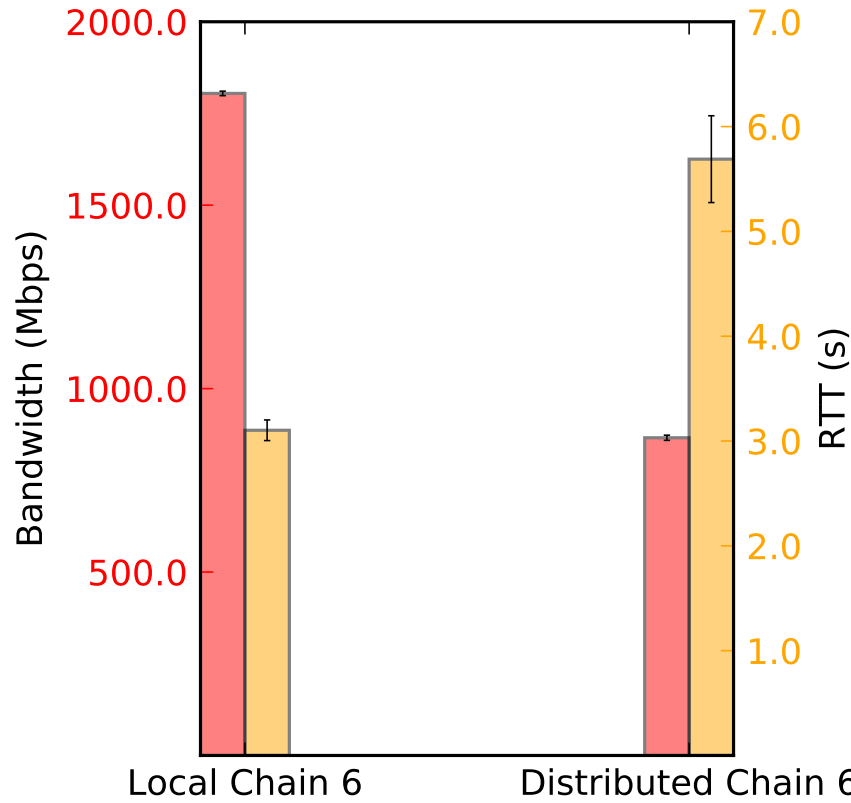


Figure 4.12: Distributed Mode: Tunnel Overhead (6 OpenWrtBB Nodes, Fixed-Range Model, Bridged WiFi Network Backend, No Link Quality Impairment)

240 seconds. Figure 4.12 shows the results. IP routing is set up along the *Chain* topology. For $x = 1$, all nodes are placed on one *Emulation Server* only. Hence, no tunnels are used at all. The examined average bandwidth is 1804.6 Mbps. The average delay is 3.1 ms. If the *Chain 6* topology is distributed among all clients ($x = 6$, one node per *Emulation Server*), the bandwidth reduces to 866.0 Mbps and the RTT increases to 5.7 ms. The lowered bandwidth can be explained by routing the traffic through 5 nodes and 5 tunnels on 6 different *Emulation Servers*. Even in this worst-case scenario, the *distributed* mode still provides sufficient bandwidth and good delay characteristics.

4.2.6 Conclusion

In this section, *MiniWorld*, a novel distributed network emulation framework, was presented. It relies on full virtualization using QEMU/KVM, offers three network backends for emulating both wired and wireless communication, and provides four mobility patterns as well as three distance-based link quality models. A novel snapshot boot mode is offered for accelerated booting of identical environments and repeating emulation runs. Connection tracking, address configuration, and network supervision features are provided for each network backend automatically. To decrease runtimes, *MiniWorld* supports distributed emulation across multiple computers, based on a resource-aware virtual machine scheduler. Experimental results have demonstrated

4.2 MiniWorld - An Emulation-based Evaluation Environment

the performance of *MiniWorld* with respect to VM boot times, network bandwidth, round trip times, and topology switching times, both for *MiniWorld*'s centralized and the distributed emulation mode.

There are several areas for future work, such as (a) providing virtual WiFi devices within QEMU instead of Ethernet devices as pseudo-wireless links, (b) developing more sophisticated models for node mobility and link quality computation, e.g., by integrating the *ns-3* simulator to support high-fidelity link emulation and by adapting the link quality depending on the used wireless interfaces, (c) investigating whether SDN switches are alternatives to Linux bridges, e.g., to provide event-driven and trace-based link impairment scenarios, and (d) integrating the QEMU-based Android emulator into *MiniWorld*, e.g., to use geographic coordinates to emulate node mobility and to evaluate location-aware Android apps.

4.3 Serval - A Robust Communication Foundation

4.3.1 Introduction

The Serval Project's [26]–[29] objective is to allow people to use mobile telephone handsets to communicate anywhere, anytime [28]. The project seeks to achieve this by creating protocols, writing software, including mobile apps, and creating complementary hardware devices that, together, are able to replicate many of the functions of a conventional cellular network to some degree (see Sec. 2.1.3). The goal is not the replacement of cellular networks, but rather provisioning the best possible set of functionality and quality of service that is feasible, without requiring any conventional infrastructure.

Currently, pilots are being planned in the Pacific region and in Outback Australia over the coming months. The Pacific trials that are sponsored by the Pacific Humanitarian Challenge²⁸ in particular will involve the provision of Serval technology to the general public in regions that are particularly vulnerable to natural disasters. It is therefore imperative that the behavior of the technology be sufficiently characterized, so that informed decisions can be made, and where any current deficiencies might exist, that they can be identified, and thus be scheduled for remediation.

In this section, an in-depth experimental evaluation is presented of the delay-tolerant networking (DTN) aspects of the Serval software stack for various network setups and usage patterns, including simulated long term use. The evaluation is based on the simulation and emulation environment called MiniWorld (Sec 4.2) to provide insights into the scenarios where Serval can be deployed with satisfactory quality and performance characteristics, without requiring the expense and complication of deploying large and potentially costly physical test networks. Since battery capacity is limited on mobile phones, a closer look at the battery drain from using Serval over various communication links, such as WiFi and Bluetooth is taken. The contributions of this research are:

- Evaluation using a hybrid simulation and emulation environment that allows us to run real OpenWRT²⁹ firmware images in an emulator, in contrast to mere simulations where only the DTN protocol can be tested.
- Various network topologies, ranging from many 1-hop neighbors and a 64-hop chain to more realistic merging islands connection schemes are evaluated.
- Several test cases mimicking common functionality, such as file distribution, messaging and peer discovery, and typical user behavior, such as rapid bulk insertion of content, writing periodic text messages, and adding different types of content every now and then, are considered.
- Different file sizes are examined to reflect different patterns of mobile phone usage, such as sharing text files (GPX data, ebooks, messages), images (map tiles, pictures), voice and video recordings (eye-witness video footage, voice memos, diaries).
- All test data, scripts and topologies are freely available and can be adapted to test other software³⁰.

²⁸<http://pacifichumanitarianchallenge.org/>

²⁹<https://openwrt.org/>

³⁰<https://github.com/umr-ds/>

Parts of this section have been published in [2].

4.3.2 Related Work

There exists a wide range of related work addressing emergency communications needs and solutions, beyond what is possible to cover in this work [49]. Nonetheless, many of the solutions in this space can be classified according to (1) the communications medium/media and modulation(s); and (2) the architectural model(s) used by each solution.

Communications media include WiFi, Bluetooth, WiMAX, GSM, TETRA digital radio, and various analog two-way and digital microwave, UHF, VHF and HF radio systems, as well as wired analog or digital systems, and satellite based systems, all available from various commercial vendors.

The architectural models can be often classified as either infrastructure-oriented, distributed (including peer-to-peer ad-hoc systems), or hybrid architectures of both approaches.

Several systems support multiple transport modalities. WISECOM [50], for example, is an infrastructure-oriented system that seeks to provide a comprehensive approach to post-disaster communications, using satellite for global connectivity and a wide range of media and modulations. A significant challenge with such systems is their overall complexity, and their dependence on a sophisticated Internet-side infrastructure.

Distinct from the transport media, considerable work has been done on designing network protocols and frameworks for emergency communications using various selections of the media and modulations listed above [51]–[53]. A resulting problem in this diversity is that interoperability can be a significant challenge and requires ongoing effort to contain and improve this situation [54], [55].

Mobile applications are also becoming more prominent in the emergency communications space [26], due to the increasing capability of modern smartphones. Several systems also employ DTN principles to mitigate the challenges that arise when forming networks from end-user devices, and without adequate supporting infrastructure [56]. Such systems are particularly relevant, due of their ability to operate when faced with the failure of infrastructure, which is a common feature in disasters and emergencies [49].

For example, FireChat³¹ is a DTN system for sending messages, but it lacks openness. Other DTN systems such as SPAN [57] and Briar³² only support specific target operating systems such as Android, and SPAN does not provide applications built on top of it. Furthermore, Forban³³ can spread files opportunistically in a DTN manner, but lacks protocol support for direct private file transfers, messaging or routing.

Liu et al. [58] have developed a DTN based mobile microblogging app for censorship resistant communication. Their focus is on the app's energy consumption in an 802.11 ad-hoc network, ignoring other means of communication such as Bluetooth or WiFi in AP mode and limiting the system to specific rooted Android devices in ad-hoc networking mode. Also, there is no support for sending large files, such as videos.

Ntareme et al. [59] have presented an approach based on Android phones using a

³¹<https://www.opengarden.com/firechat.html>

³²<https://briarproject.org/>

³³<http://www.foo.be/forban/>

4 Disruption-tolerant Device-to-Device Emergency Communication

store-and-forward architecture. Services such as email are transparently delivered via DTN, but the solution requires special server software in addition to the Android app. Energy and bandwidth consumption were measured, but scalability and performance in different scenarios were not evaluated.

Heimerl et al. [60] attempt to solve the problem of poor cellular coverage and power outages in rural areas by using low-cost GSM hardware and a system for reduced power consumption. While this approach is interesting for feature phones and services such as voice calls and text messages, it still requires infrastructure to function.

4.3.3 Experimental Evaluation

In this subsection, the details of the evaluation setup are described including the selected network topologies and tasks. Afterwards, the results of our evaluation are presented.

Experimental Setup

The experimental setup for the in-depth evaluation of Serval, including the hard-/software environment used, the parameters measured, the network topologies chosen, and communication scenarios, is presented below.

Simulation/Emulation Environment To evaluate the performance in a realistic manner, the *MiniWorld* network emulator presented in Section 4.2 is used. This gives us the opportunity to use the OpenWRT build chain for building router images that include Serval. OpenWRT is also used on real world routers such as TP-Link MR3020 or the Mesh Extenders of the Serval Project. Having a full operating system with its own network stack running on each node gives a much better picture of real life performance than pure protocol simulation.

All tests are performed on a 64 core AMD Opteron 6376 CPU with 256 GB RAM, simulating up to 100 virtual nodes, each one with 512 MB RAM and 2 GB of storage space. These quite limited values allow us to investigate how Serval performs on older smartphones like the original Samsung Galaxy S or similar, which are common in developing countries.

Measurements Standard Unix tools are used to measure system properties, with a measuring interval of one second. For memory consumption, CPU and I/O usage *pidstat*³⁴ is used to monitor the statistics of the Serval process from within a node. Disk space is measured with *du* and *df*, both from the GNU coreutils³⁵. Network usage is measured on the *MiniWorld* bridge interfaces of the host system using a custom Python tool³⁶ based on *libpcap*³⁷. Insertion points in time for the Rhizome store are derived directly from Serval's log, while the general file count is logged using direct *servald* calls.

³⁴<http://sebastien.godard.pagesperso-orange.fr>

³⁵<http://www.gnu.org/s/coreutils/>

³⁶<https://github.com/umr-ds/serval-tests/blob/master/netmon.py>

³⁷<http://www.tcpdump.org>

Table 4.1: Network Topologies

Name	# Nodes	Description
Hub	48	All nodes connected to each other
Chained	64	Pair-wise connected
Islands	100	Partitioned islands, merging over time

Network Topologies Several network topologies are studied, as shown in Table 4.1. The *Hub* topology connects 48 nodes with each other. It represents a scenario with a high number of direct neighbors all using bandwidth, flooding each other with status information and new files, sharing the same transport channel. Typically, the number of direct neighbors is limited by the radio range of WiFi or Bluetooth (i.e., often less than 48). Thus, *Hub* is challenging for Serval and also the radio link itself.

The *Chained* topology consists of a chain of 64 nodes, thus the last node is 63 hops away from the first node. Typically, network connections over the Internet require less than 16 hops. In a delay-tolerant mobile mesh network, more hops might be needed for messages to reach their destination compared to static networks physically optimized for minimum hop numbers and maximum throughput.

The *Islands* topology represents a partitioned network that slowly merges over time. At the beginning, there are 100 nodes in small islands with only a few neighbors. Between these small islands there are no links, but after a predefined time a few of them merge together, exchanging all their information that they have collected so far. Finally, there are two big islands where one node acts as a bridge between the two, and all accumulated data from one island has to pass through this node to propagate to the other island.

All topologies are used in two configurations, one modeled after the common 802.11g standard with a 54 Mbit/s limit on each link and one with no bandwidth limitations.

Scenario Tests Based on these topologies, several tests were designed, as shown in Table 4.2.

Idle (I) simply starts Serval and waits until all nodes have found each other. This test serves to evaluate how long the discovery phase takes in various network setups and how much traffic Serval produces while idling.

Mass Files (MF) pre-generates a number of files and inserts them at one specific node. The goal is to evaluate whether Serval can handle a large number of files at once. Propagation through the network is observed to reveal problems related to high bandwidth, storage and/or CPU usage.

Mass Messages (MM) is designed to test the messaging subsystem of Serval by flooding the network with text messages. A number of messages is sent at once to every single node in the network no matter if it is currently reachable or not.

Periodic Files (PF) is designed to observe the long-term behavior of the system. Files are added at random points in time by every node. A real world analogy is: people taking pictures occasionally and sharing them with everybody else.

Periodic Private Files (PPF) is a special case of *PF* where files are not shared with the public but sent to a randomly chosen recipient.

4 Disruption-tolerant Device-to-Device Emergency Communication

Table 4.2: Scenario Tests

Name	Short	Description
Idle	I	Node discovery, no actions triggered
Mass Files	MF	Insert bulk of file set at once
Mass Messages	MM	Insert bulk of messages at once
Periodic Files	PF	Periodic adding of files
Periodic Private Files	PPF	Periodic adding of private files
Periodic Messages	PM	Periodic sending of messages
Combined	C	All periodic tests together

Table 4.3: Test File Sets

Name	Sizes	Description
Small	64K, 256K, 512K	Small pictures, map data, text files
Medium	1M, 5M, 10M	Camera pictures, audio recordings
Large	25M, 50M, 100M	Recorded video
Mixed	all of the above	-

Periodic Messages (PM) is designed to evaluate the Serval messaging subsystem. These messages are also directed to a specific recipient and are not meant for the public.

Combined (C) is designed to run all periodic tests (*PF*, *PPF*, *PM*) at once. Similar to real life situations, the nodes change their behavior and there is a competition for the resources in the network. Broadcasting files, sending files to “friends” and writing text messages all have different requirements.

Data Sent Text messages consist of a fixed string plus a timestamp in milliseconds when a message was sent. Since these messages are meant to mimic real world chat, the total string length is kept small (53 characters). According to a chat study of Battestini et al. [61], text messages sent by males had an average length of 47 characters and for females 58 characters.

Files have different file sizes representing different types of data, as shown in Table 4.3. The *Small* file set contains randomly generated files ranging from 64 KB to 512 KB; large text files, ebooks, small pictures or other data such as map tiles typically have these sizes. In the *Medium* file set we have files between 1 MB and 10 MB, which is nowadays the size of pictures taken with mobile phones or some audio recordings. Recorded video or software bundles are represented in the *Large* file set and are generated in the range from 25 MB to 100 MB. Finally, there is a *Mixed* file set where small, medium and large files are included.

Test Execution All file related tests were performed with all four file sets, every test was executed on all topologies with limited and unlimited bandwidth resulting in a

total of 114 tests. While some tests (e.g., *MF*) are count-based and terminate after every node has received a specific number of files, other tests (e.g., *PPF*) are time-based - always running for the same duration. Each test was performed 5 times, resulting in a total of 570 test runs.

Experimental Results

In the following, various results regarding Serval's behavior during the experiments are presented.

Idle Behaviour To investigate the idle behavior of Serval, we looked at network traffic, CPU load and memory usage after the initial discovery phase, without triggering further actions. In every scenario, whenever Serval is started, there are peaks in the network load, in the *Chained* and *Hub* topologies at approximately 10 to 12 Mbit/s. After this peak, *Chained* has a summed average network traffic of around 0.7 Mbit/s, whereas the nodes in *Hub* produce 6 Mbit/s. This behavior is caused by Serval's information distribution strategy, because it announces status information, such as the list of files in Rhizome, periodically via broadcasts. Since there are 47 neighbours for each node, traffic is relatively high in the *Hub* topology. *Islands* has extrema whenever partitions merge. The traffic during peaks grows with the number of nodes.

CPU usage of the Serval process correlates with network load in our scenarios, but never gets larger than two percent per node. Serval uses around 4 MB of memory in all scenarios.

Moreover, the discovery time of each topology is different. For *Hub*, the average time of a full network discovery is approximately 5 seconds, since every node has a direct connection to all others. In contrast, the *Chained* topology takes about 20 seconds, because announcements have to be forwarded through all other nodes.

In some experiments, Serval's address abbreviation (Sec. 2.1.3) mechanism caused conflicts under special circumstances, depending on the keys and when different nodes announce themselves for the first time. If a node already has seen another node with the same abbreviated address, it is ignored, potentially causing a partitioning of the network. To circumvent such effects, we modified Serval to generate unique prefixes for the desired node number in our tests.

Hub Constraints For *Hub*, a single bridge interface was used to connect all nodes. Since each node is a single hop away from all other nodes and Serval uses broadcast packets to announce meta-data (e.g., the files of a node), each node is flooding all neighbors with this information. Since the number of adjacent nodes affect the CPU consumption of the respective node, in the *Hub* topology the CPU usage is always higher than in the corresponding test in *Chained* or *Islands*, due to the high number of direct neighbors.

Topology Characteristics Fig. 4.13 shows *Mass Files* tests with a *Mixed* file set in different topologies. It shows how transfer rate in Mbit/s, size of the Rhizome database and the CPU usage change over time. The transfer rate is stacked for all links. The Rhizome size is the stacked database sizes of all nodes.

4 Disruption-tolerant Device-to-Device Emergency Communication

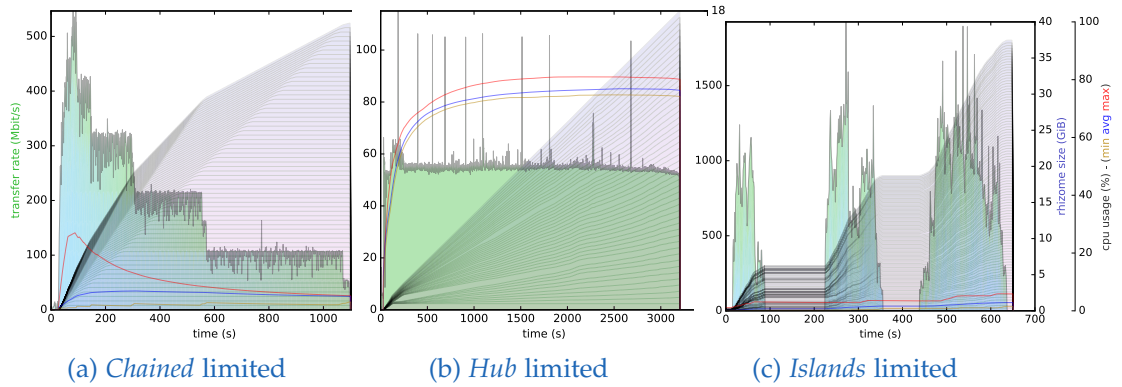


Figure 4.13: MF Mixed: Rhizome store size, network and CPU load

Fig. 4.13a shows a limited (802.11g) *Chained* topology, in which five phases are visible, caused by the Rhizome prioritization based on file sizes. Small files are delivered first and therefore can be distributed earlier by the following nodes. The bigger the files get, the less total network utilization is achieved. Despite this effect, a constant stable data flow is visible, and the Rhizome store grows constantly. The maximum CPU load correlates with network usage, since the most active network nodes do have the highest CPU usage.

In Fig. 4.13b, a limited *Hub* topology is shown. Though a constant 54 Mbit/s data flow is visible, the spikes exceeding 54 Mbit/s are measurement errors, caused by differing network backend and traffic monitoring timers. With a constant network load caused by the file transfers, the disk usage also grows linearly as expected in this case, meaning that the network load is not dominated by status and management information but real content distribution. Compared to Fig. 4.13a the average CPU usage is about 10 times higher, as explained in Sec. 4.3.3.

For *Islands*, CPU usage increases every time the network changes. Looking at *Periodic Files* tests, the max. CPU load rises to 15% when large files are inserted, since they have to be redistributed among the other nodes. Fig. 4.13c shows the *Mixed* file set in MF, which peaks at around 7% CPU load. Since many of the files already exist on various nodes, every time new network connections are set up, the impact on the CPU is relatively low compared to *Hub*. In general, smaller files have a negligible impact on the CPU.

The *Periodic File* tests with small sizes do not show any unexpected behavior in terms of CPU consumption in *Chained*, the CPU peaks at about 10%. When the files are encrypted as in *PPF*, the CPU utilization is slightly higher, at about 15%, due to CPU intensive cryptographic operations.

The file size influences CPU utilization, which greatly impacts the inserting node. For instance, when sending *Small* files in *Chained*, there is no significant change of CPU utilization compared to idling, whereas file set *Large* utilizes the CPU up to 35%. Bigger files lead to more time consuming hashing, as it is required by the corresponding protocol. Thus, every node receiving the file needs to compute a hash, verify and redistribute it, which also leads to a higher load.

In terms of CPU usage, *Islands* is a combination of *Chained* and *Hub*. CPU usage does not exceed 50%, since the total number of neighbors per node is not as high as in *Hub*.

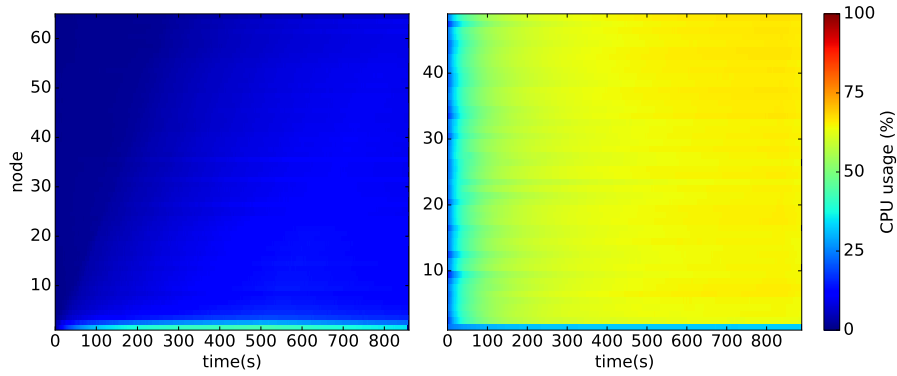


Figure 4.14: MM CPU usage over time. Left: unlimited *Chained*, right: unlimited *Hub*.

In the message based tests, the measured CPU consumption correlates with the number of messages sent. For MM, the behavior differs depending on the topology used. Fig. 4.14 shows the CPU usage per node of two experiments over time. Using *Chained*, the inserting node peaks at 30% CPU load compared to the receiving nodes, which consume about 15%. Using *Hub*, the load of the inserting node remains the same. In contrast, the receiving nodes constantly consume about 65% CPU. *Hub* suffers from the broadcast overhead (Section 4.3.3), but this does not fully explain the high load, as the sending node is not affected. Further investigating this behavior, it can be tracked back to recurring hashing and encryption in Rhizome Journal syncing, which is the core of MeshMS messaging.

PM results differ from MM. For *Chained*, the CPU utilization is relatively low at about 15% maximum. This correlates with the CPU load of the non-inserting nodes in MM. Since they are added periodically, the CPU overhead is negligible here. *Hub* behaves differently than in the file based tests or MM: The *PF* tests show that in every topology the more files are injected in the network, the more CPU is needed to handle the broadcast packets. Messages are not announced further after reaching their destination and being acknowledged by the recipient. The obvious consequence should be that the CPU usage decreases. However, as indicated by Fig. 4.15, once the CPU peaks at about 25%, it does not settle any more, but increases even further, although the network load decreases to the idle level and the Rhizome database size is at its maximum, which indicates that all messages have arrived. This behavior cannot be transferred to *Islands*, where the inserting nodes peak at about 40% and all other nodes do not exceed 15%.

For *C* tests, the general CPU usage is similar to other file based tests. The only difference is the fact that in *Chained* and *Hub* the CPU usage increases by 5% after about 500 seconds and also correlates with the network load, similar to the behavior depicted in Fig. 4.15. This problem emerges when sending messages over a longer time period. Since *Islands* is not in the final state at the beginning of the test in terms of the links between the nodes, this result can not be observed in this particular topology.

Network Performance One goal was to test to what extent Serval is able to use available bandwidth. *Chained* was created to assess this.

The cumulative transfer rate using Rhizome in this topology reached 500 Mbit/s to 2 Gbit/s, depending on the file sets, with *Large* being the fastest. That is, up to 2 Gbit/s of traffic was being carried over the set of hops in the chain, with each seeing an

4 Disruption-tolerant Device-to-Device Emergency Communication

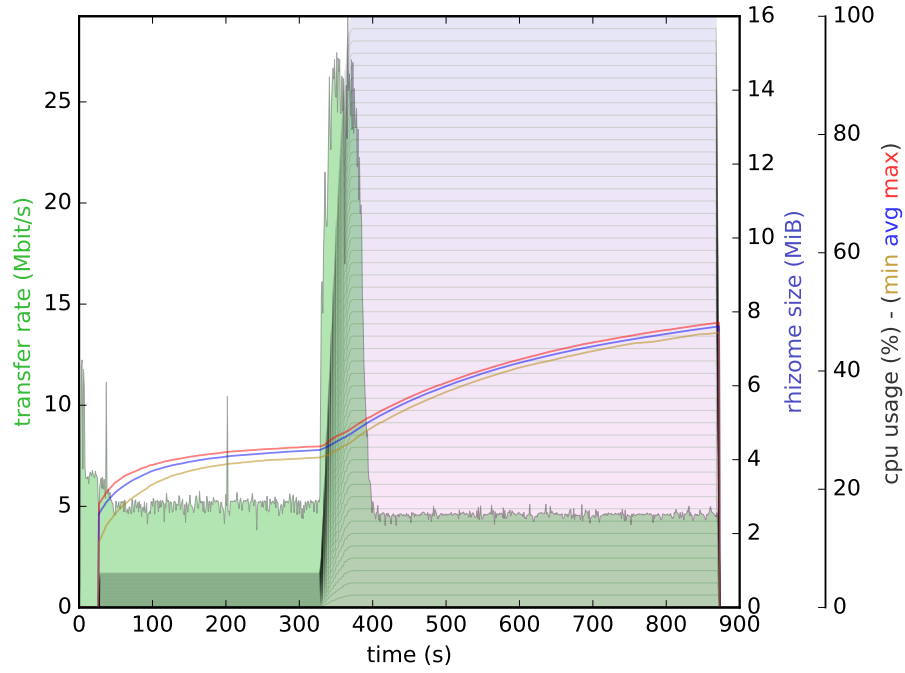


Figure 4.15: *Hub* limited *PM*: Rhizome store size, network & CPU

average utilization of 32 Mbit/s. Tests that transfer large files over an unlimited network show that Serval is able to use even more bandwidth, since the highest measured transmission speed from one node to another can be up to 160 Mbit/s.

Using *Chained*, the hop-to-hop transmission time can be modeled, since node n is able to receive a file just after node $n - 1$ received it. Fig. 4.16 shows the hop-to-hop transmission times of the *Medium* file set. The five files of each size are grouped into one box plot, while the colors present five different runs of each experiment. The median transmission times for 1, 5 and 10 MB files are 0.54, 1.06 and 1.85 seconds, and only 0.27 sec for 64 KB files. From these values, a simple correlation for the transmission time can be derived: $T(size_{MB}) = 0.16 \cdot size + 0.26$, which also holds for the *Large* set. The formula indicates a net transmission rate of around 31 Mbit/s, with a 0.26 sec delay.

The average speeds are lower, because files are exchanged node-by-node, and can only be spread to node $n + 1$ after reaching node n , resulting in an effective end-to-end bandwidth, for a given bundle, inversely proportional to the number of hops. This compares favorably with end-to-end ad-hoc wireless routing protocols, where the effective end-to-end bandwidth drops by approximately half for each additional hop.

Briefly considering the different topologies, the network utilization in *Islands* for file based tests is generally about the same as in *Chained*, since each node has only a few neighbors, in contrast to *Hub*, which is always able to saturate all links due to the high degree of connection among nodes.

Messages in Serval are effectively transported as small files, with a payload size of 53 bytes in both *PM* and *MM* cases. The network load shows a behavior similar to small files in the *PF* test, peaking at up to 40 Mbit/s at all topologies and regardless if the network is limited or not.

The network load for *C* tests in all topologies is similar to the file based tests, independent of bandwidth limitations. The only difference is the increase of the network

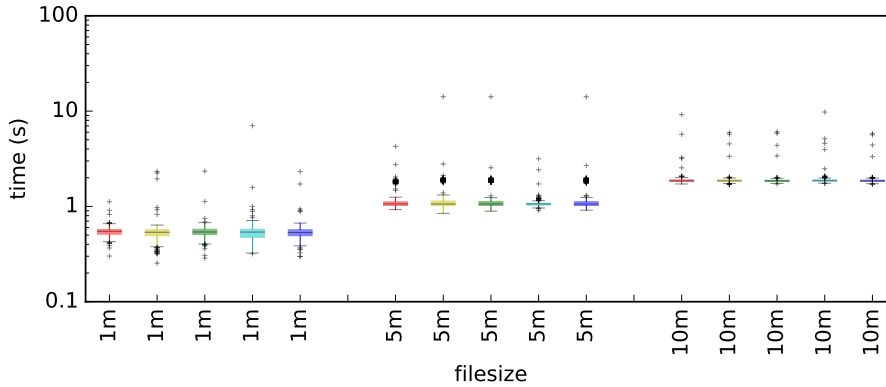


Figure 4.16: *Chained* limited *Medium* file set: File-size-grouped hop-to-hop delivery periods of five runs.

load after about 500s on *Chained* and *Hub*, as shown in Section 4.3.3.

In *Hub*, small files take between 1 and 4 min to arrive on the last node in the limited network links. This increases linearly, up to 20 min, with increasing file size. If the network is unlimited, transmission time reduces to between 18 sec and 9 min, depending on the file size. One difference between *Hub* and *Chained* is the runtime. *Small* files are transmitted faster in *Hub*, whereas *Large* files are faster in *Chained*. The time overhead for file announcements is relatively higher for *Small*. Even with a lower total bandwidth (*Hub*: 54 Mbit/s for 48 nodes vs. *Chained*: 54 Mbit/s pairwise), *Hub* can achieve faster transmission rates. The limitation of network speed does not influence this behavior, only the overall transmission time increases.

The transfer times of messages differ from topology to topology. While it takes about 350 sec in *Chained* until all messages arrive at their destinations, it can take up to 900 sec in *Hub*. This again shows that the high number of 1-hop neighbors in *Hub* is challenging for Serval. The transmission time for messages in the *C* tests depends highly on the used file set, rather than on the topology and network speed. The reason is that the network is saturated with big files, which leads to overall higher transmission times for messages.

Energy Consumption The *Idle* test in Section 4.3.3 showed network peaks caused by Rhizome status information announcements. Therefore, the energy consumption of the announcements is evaluated: Two devices send announcements in different intervals. Fig. 4.17 shows the energy consumption for peer A using different announcement intervals at peer A and peer B. With a 0.5 sec or 1 sec interval, the consumed energy is 9% higher than in idle state. With a 2 sec interval, the consumed energy is only 3% higher than in idle state. With a higher interval of 4 sec or 8 sec, only negligible decreases in energy can be achieved.

Furthermore, the power consumption during *MM* and *MF* tests were evaluated. Two peers were connected via an 802.11n WiFi Access Point. Peer A inserts files and messages into Rhizome in the same manner as *MM* and *MF* tests. The power consumption of peer B, a Raspberry Pi 3, is then measured with the Odroid Smart Power measurement device, an external power meter. The aim of these experiments is to measure the energy overhead for running Serval on a device, which allows conclusions about the power drain of Serval on battery-powered devices.

Fig. 4.18 shows the power consumption during different Rhizome file set insertions

4 Disruption-tolerant Device-to-Device Emergency Communication

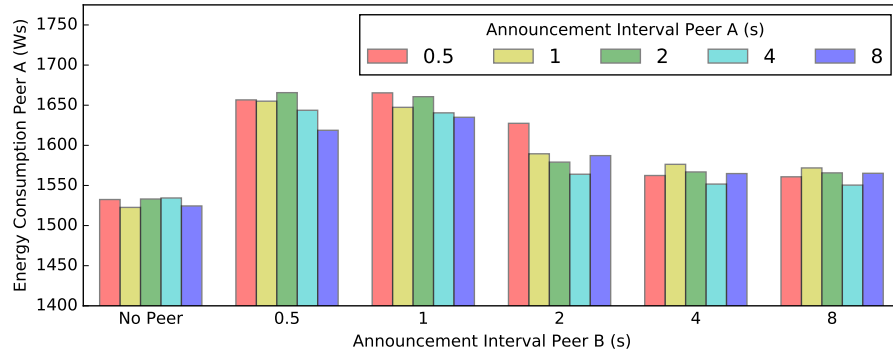


Figure 4.17: Energy consumption of announcement intervals

similar to *MF*. The file sizes are increased during the phases f_1 - f_4 . During f_1 , the file sizes are smaller than 1 MB, resulting in a negligible additional power consumption. The bigger the transmitted files are, the more power is consumed. The comparison between receiving files and sending files shows an unexpected behaviour: In all phases f_1 - f_4 , sending files is less expensive compared to receiving files, on the average between 0.05 and 0.1 W (3-6%). This counter-intuitive result is caused by additional CPU consumption of the Rhizome checksum calculation during reception. Compared to a 1.53 W mean idle value of peer B, the power overhead introduced by Serval is between 0.01 and 0.13 W (1-8%) during phases f_1 - f_4 .

In another experiment, the power consumption during different message insertions similar to the *Mass Messages* test was measured. The results show a power consumption peak between 1.81 and 1.91 W during a short period of reception, followed by a phase of negligible additional power consumption. During the reception of 100 messages, a mean value of 1.69 W (10%) additional power consumption is measured.

A better energy efficiency during message transmission could be achieved by using Bluetooth. It consumes a significant amount of energy during device discovery, but has a lower power consumption during data transmission than WiFi. Due to the low energy efficiency (joule per bit) of Bluetooth compared to WiFi, it consumes significantly more energy for large data transmissions. During an experiment, we measured a 32 times better energy efficiency of WiFi compared to Bluetooth for files between 512 KB and 16 MB.

4.3.4 Conclusion

In this section, an in-depth experimental evaluation of the delay-tolerant aspects of Serval for various network setups and usage patterns was presented. The results show satisfactory performance of Serval when deployed in partitioned scenarios and extreme examples of network topologies. Furthermore, Serval's energy consumption was evaluated, having the limited battery capacity of mobile devices in mind.

In particular, the experiments indicate that there is a sweet-spot for the trade-off between up-to-dateness and energy consumption regarding announcement intervals. Furthermore, Serval can handle a realistic number of files over a longer time period. In the *Chained* topology, neither the CPU load nor the used network bandwidth leads to out of service situations. All tests with the *Hub* topology show that in a highly used network the announcements consume a considerable portion of the available

4.3 Serval - A Robust Communication Foundation

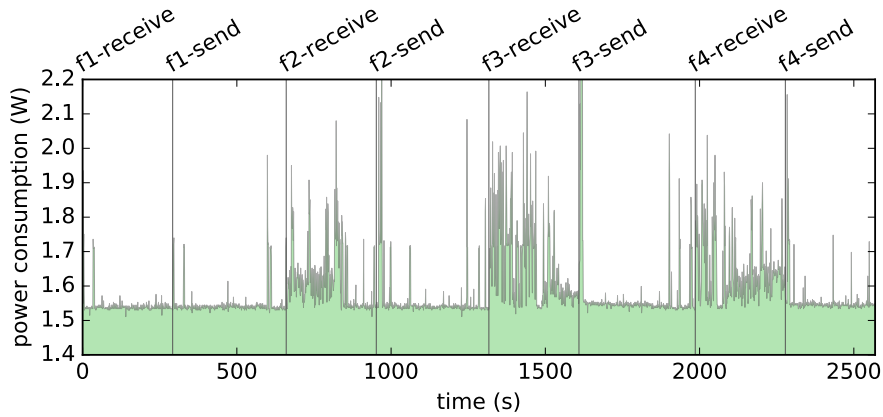


Figure 4.18: Power consumption during different Rhizome file set insertions (*f1-f4*) similar to the *Mass Messages* test.

bandwidth. In emergency situations or in long-term setups this could have a negative effect depending on the number of people in direct communication range. The *Combined* tests in our *Islands* topology demonstrate that Serval works flawlessly in adapting to heterogeneous environments where users have different requirements at the same time and the topology changes over time.

There are several areas for future work. Mobility simulations should be carried out, preferably with real world movement patterns gathered from past events. More powerful hardware with higher numbers of nodes should be used to run the simulations and emulations, to further investigate Serval's scalability properties, particularly in highly-connected topologies, like *Hub*. The defect that has been exposed in the address abbreviation code should be rectified. An evaluation of Serval's non-DTN related features, such as voice calls, could further increase the attractiveness of Serval as a solution for emergency or off-grid communication.

4.4 Optimizing Epidemic Announcements

4.4.1 Introduction

As shown in the previous section, several network protocols rely on nodes broadcasting announcements to other nodes. Examples include service discovery (*Bonjour/ZeroConf*, *Samba*), routing algorithms (*RIP*, *OLSR*), and peer-to-peer or delay-tolerant networking (DTN) systems (*Forban*³⁸, *Serval*³⁹). While the traffic generated by periodically sending announcements might be negligible in wired networks with high-speed links, bandwidth in wireless networks, such as 802.11, Bluetooth or various mobile ad hoc networks (MANETs), is precious and limited. For example, spontaneous smartphone networks become more and more important not only by providing pervasive wireless Internet access during large human crowd gatherings, but also during emergency situations or post-disaster recovery [62].

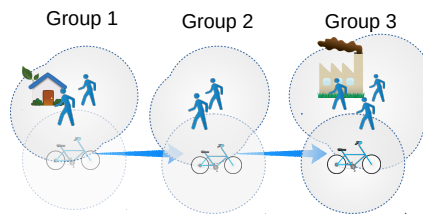


Figure 4.19: Drive-by store-and-forward data exchange.

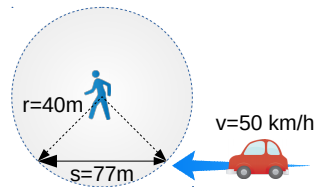


Figure 4.20: Drive-by window of opportunity example.

In an emergency communication scenario, the main goal is to spread messages and files produced at a disaster site fast among reachable nodes. Therefore, data is passed around in an epidemic fashion to as many neighboring peers as possible. Typically, some nodes are more static, such as devices of people trapped in their houses or small emergency camp sites forming islands, while other nodes are on the move (by bike, car, foot), which by passing through these islands act as carrier-pigeons to distribute information further (see Fig. 4.19). These islands have a higher density than typical sensor networks. To make optimal use of the short time in case of a drive-by, it is important to find a peer for data exchange very fast. Since any peer can initiate data synchronization, a special treatment of mobility is not necessary. Depending on the used wireless technology, a mobile phone might have an effective range of 14-80 meters to communicate with others. Thus, if we assume a WiFi radius of 40 meters and a static node being 10 meters away from a street, a car driving on the street would be

³⁸<http://www.foo.be/forban/>

³⁹<http://www.servalproject.org/>

in the WiFi range for about 77 meters (see Fig. 4.20). The car passing by, assuming it moves at about 50 km/h, would have just under 6 seconds for node discovery and exchange of data. This is plenty of time for transferring, for example, two 6 megapixel pictures and setting up connections via a standard 54 Mbps link. Therefore, for the fast moving node, one of the more static peers is sufficient to start a data transfer. Since all information gets replicated in this scenario, the fast moving node does not need to know all possible neighbors. The static node can distribute the data further among its neighbors. Discovering *all* direct peers as fast as possible is neither necessary nor beneficial for the static nodes. Under these assumptions, it is reasonable to use dynamic announcement intervals instead of the typically used static announcement intervals. Furthermore, dynamic announcement intervals require not only less network resources, but also potentially save more battery capacity than static announcement intervals.

In this section, several approaches are presented to realize dynamic announcement strategies that facilitate fast reception from at least one other node while trying to keep the overall communication overhead as low as possible. Experimental results in terms of performance properties and energy consumption are given to illustrate the benefits of dynamic announcement intervals in wireless on-demand networks. In particular, this section makes the following contributions:

- Various strategies for realizing dynamic announcement intervals optimized for different network setups are presented.
- An experimental evaluation of all proposed strategies, including static and random announcement strategies, with respect to bandwidth usage, announcement distribution and energy consumption is presented.
- Test environments suited for various topologies, such as large stable networks, islands merging and networks splitting, are investigated.
- The results are directly applicable to local peer-to-peer content distribution systems in emergency scenarios, such as *Forban* and *Serval*.

Parts of this section have been published in [3].

4.4.2 Related Work

There are several publications that investigated problems associated with static announcement intervals in various protocols and application scenarios.

Natsheh et al. [63] proposed a solution based on fuzzy logic to optimize hello messages in dynamic ad-hoc routing. Their work focused on the mesh routing use case, and experiments with a maximum of 35 simulated nodes were presented. Furthermore, Khalaf et al. [64] investigated the broadcast storm problem in mobile ad hoc networks. The authors presented a probabilistic approach to improve the situation in a mesh routing scenario.

Ahmed et al. [65] addressed the problem of beaconing in vehicular ad hoc networks (VANETs). Combinations of controlling a beacon's transmission power, transmission rate, and contention window at the MAC layer were proposed to achieve efficient beacon communication in VANETs. Another approach devoted to improve the problems related to static beaconing intervals in ad hoc networks was presented by Tahar et al. [66]. Hess et al. [67] investigated peer discovery in mobile opportunistic networks by considering the mobility of nodes.

4 Disruption-tolerant Device-to-Device Emergency Communication

Peng [68] proposed an adaptive mobility-aware MAC protocol for wireless sensor networks. Apart from optimizing the number of messages, the energy consumption was investigated. Lim et al. [69] presented an approach called RandomCast to improve the energy efficiency of 802.11 ad hoc networks. In this approach, the sender can specify the desired level of overhearing of neighboring traffic, trying to find a balance between energy consumption and routing performance.

Using perfect difference sets for neighbor discovery, Link et al. [70] presented an energy efficient approach for wireless networks. The authors focused on sensor networks and DTNs with sporadic communication, whereas we focus on networks with higher communication frequencies in local clusters.

Peer-to-peer content distribution is another scenario where announcements are relevant, and a trade-off must be made between central tracker-based peer discovery and distributed peer discovery. Dán et al. [71] presented a hybrid approach that uses individual trackers and a gossip protocol to improve peer discovery. By hopping between swarms and redistributing known peers, efficiency is increased.

Liu et al. [58] developed a censor-ship resistant delay-tolerant network for message exchange and evaluated it with respect to performance and energy consumption. To avoid energy draining broadcasting with fixed intervals, the authors adopted an approach presented by Zheng et al. [72] based on asynchronous wake-ups for ad hoc networks. Another delay-tolerant networking system designed specifically for data synchronization in emergency situations was presented by Paul et al. [73]. While optimizations are proposed to speed up file transfers and syncing, the actual peer discovery was realized by simple broadcasts with fixed announcement intervals.

During an experimental evaluation of *Serval* as a delay-tolerant emergency communication platform, Baumgärtner et al. [2] (see also Section 4.3) found that regular broadcasts used for node discovery or announcements of routing and data storage information especially in networks with many direct peers require high network bandwidth. The study showed that around 2 seconds of announcement delay was the best trade-off between quick peer discovery and conserving energy with the stock implementation made available by the Serval Project.

By exploiting social network characteristics for assisting ad hoc peer discovery, Zhang et al. [74] attempted to find optimal beacon probing rates with constant intervals for each group of users. As stated by Wang et al. [75], peer discovery itself can be as energy consuming as making phone calls.

Trifunovic et al. [76] presented a solution for opportunistic networks of stock mobile devices using 802.11. Since ad hoc mode and Bluetooth pairing does not really work in practice on current mobile devices, open access points and intelligent switching of clients between these access points were used.

While most of the mentioned work is highly specific to the studied use cases, the general picture is that adaptive or dynamic announcement intervals usually outperform static ones, not only with respect to network performance, but also regarding energy consumption. In this scenario, a small dense clusters of nodes is considered, where a few nodes act as mobile bridges between these islands, in contrast to most sparse sensor networks. Furthermore, most approaches focus on lower layer technologies, whereas these algorithms here can be applied on the application layer without operating system support.

4.4.3 Design

Dynamic Announcement Intervals

In this section, several dynamic announcement strategies, the constraints associated with them, and quality properties to evaluate their performance are presented.

Announcement Strategies

Several novel strategies for realizing dynamic announcement intervals have been developed. Each strategy has access to the current announcement delay, the global number of announcements seen at the last observation interval and the current number of unique peers. The strategies are described in the following:

Static The *Static* announcement strategy is the basic announcement approach used by most current broadcast protocols. There is a fixed interval defined for every node in which an announcement is sent. This also means that the generated global traffic is growing linearly with the node count. By default, this interval is set to a 2 second delay in our tests, which also is the recommended value for MANET NHDP [77].

Random In the *Random* strategy, every node chooses a random announcement delay. This delay is a random number between a minimum and a maximum (as described in Section 4.4.3) for every observation interval. The distribution of the random numbers, depending on the network size, heavily influences the performance of this strategy, as well as the duration of the observation interval.

RandomSweet In this strategy, *Random* is extended. The announcement interval is only set randomly if the current global announcement rate is higher than one announcement per second or less than the minimum number of announcements per second (see Section 4.4.3). Thus, if the network has reached a stable state, this strategy does not change anything and sticks to the last randomized delay for each node. This stabilizes the network if by chance optimal delay combinations are found, at least until nodes join or leave the network.

Step After every observation interval, the *Step* strategy checks the global announcement count. If the count is higher than one announcement per second, the node's announcement delay is increased by one second. If the count is lower than 0.5 announcements per second, the node's announcement delay is decreased. This leads to gradually narrowing down to a most suitable announcement delay over time.

StepRand In this strategy, *Step* is extended by adding randomness to each step. While the conditions remain the same as in *Step*, a random value between 0 and 0.5 seconds is added or subtracted to the announcement delay.

MaxFirst *MaxFirst* is a rather defensive strategy: whenever a high global announcement rate is detected (more than one announcement per second), the node's announcement interval is set to the observation interval, i.e., the maximum possible announcement delay is tried first, hence the name. Then, if less than 0.5 global announcements

4 Disruption-tolerant Device-to-Device Emergency Communication

per second are present, the strategy decreases the announcement delay by one second per iteration, until the local minimum of 0.5 seconds is reached. Thus, a very low announcement frequency is favored, which should be beneficial in larger or fast growing networks.

MinFirst *MinFirst* reverses *MaxFirst*, and thus is an aggressive announcement strategy. Whenever less than 0.5 announcements per second are detected globally, the announcement delay is set to the local minimum of 0.5 seconds. Otherwise, the announcement delay is increased by one second per iteration, until the observation interval is reached. This strategy supports scenarios where most of the time only very few peers are in direct vicinity of each other.

Unsteady In the *Unsteady* strategy, each announcement delay is computed only on the basis of the number of unique peers known by a node and not on the global announcement rate like in the other algorithms. The goal is to reach a global rate of one announcement per second. Looking at the current peer count, an announcement interval is computed to complement the announcement intervals of the other nodes. Using this method, the strategy should be able to adapt to new situations as fast as defined by the observation interval.

Constraints

To guarantee that a node can be discovered, an *observation delay*, with the same value for all nodes, is defined. This is the time between re-evaluation and before another change in the announcement frequency can happen. Each node has to announce itself at least once per observation interval. All nodes must set the announcement delay after the observation delay is over. This enables a better comparability between the announcement strategies.

The *observation delay* is set to 20 seconds in all experiments, since the baseline for static announcements is 2 seconds. Therefore, it is reasonable to re-evaluate the situation after 10 standard announcements. The higher the delay, the longer it takes for the network to adapt to new situations. A very short delay in conjunction with the premise that each node should at least send one announcement per interval leads to higher loads, especially with higher node numbers. Thus, a delay of 20 seconds ensures that within this interval *all* peers in the direct neighborhood are discovered.

Quality Properties

To evaluate and compare different strategies for dynamic announcement intervals, universally applicable quality properties must be defined. The main goal is to globally have one announcement per second at any given time, not less, but also not much more to conserve resources. This goal is motivated by the drive-by scenario described in Section 4.4.1, in which 10% of the window of opportunity would be used for peer discovery under this assumption.

Global Announcement Rate The *Global Announcement Rate* is measured by counting the announcements per second. This parameter is the main optimization goal for the algorithms, since it is directly correlated with the bandwidth used for peer discovery.

Global Announcement Gaps The *Global Announcement Gaps* are measured by the time periods between two announcements. The *Global Announcement Gaps* are important to observe, since they reveal how long a new peer needs until it receives an announcement from the rest of the network. Although this value is roughly the inverse of the *Global Announcement Rate*, its distribution can reveal other aspects, as observed in our experiments.

Adaptation Rate The *Adaptation Rate* represents the time needed for an announcement strategy to adapt to a new situation. It describes the situation that all nodes are started at the same time, and defines the moment when no significant change in the number of announcements is recognizable.

4.4.4 Implementation

In this section, implementation issues of the announcement strategies and the network using them are discussed.

Mesher

To investigate dynamic announcement intervals, a simple broadcast service was extended to provide easily exchangeable announcement algorithms for peer discovery. *Mesher*⁴⁰ is a simple local chat written in Google's *Go* language by me, and therefore is easily extensible. It utilizes broadcast packets for neighbor discovery and for exchanging public chat messages. *Mesher* uses a static announcement interval of 2 seconds in its default configuration, and thus the network traffic is growing linearly with the node count. Each announcement contains the elliptic curve public key of the sending node, the services provided by the node, 512 bytes random data to simulate database states and a cryptographic signature, resulting in 642 bytes per broadcast packet. Other protocols might use larger or smaller announcement packets, depending on the type of state that is broadcasted.

Dynamic Interval Computation

To evaluate various interval computation methods including dynamic changes, the corresponding algorithms needed to be easily exchangeable. Therefore, the algorithms are implemented using an embedded *JavaScript* engine, and an interface was defined to hand over useful information to access it in the main *Go* binary:

- `get_announce_count()`
- `get_and_reset_announce_count()`
- `get_peer_count()`
- `get_announce_delay()`

After analyzing the provided values, the algorithms are able to set a new announcement interval using `set_announce_delay(Int)`.

⁴⁰<https://github.com/gh0st42/mesher>

4 Disruption-tolerant Device-to-Device Emergency Communication

Announcement Strategies in Mesher

For all announcement strategies, the same template (see Listing 4.1) in *JavaScript* is used where one specific function is responsible for computing any changes. Each strategy gets the current announcement delay and the global number of announcements seen in the last observation interval. This setup proved to be perfect for rapid prototyping of new algorithms without recompilation or modifications of the main binary.

Listing 4.1: Basic layout of the announcement strategies

```
var observation_interval = 20000;
var total_count = 0;
var min_delay = 500;

set_announce_delay(2000);
for(;;) {
    sleep(observation_interval);
    var cur_count = get_and_reset_announce_count();
    var cur_delay = get_announce_delay();

    // call scheduler and set new delay there
    scheduler(cur_count, cur_delay);
}
```

4.4.5 Experimental Evaluation

In this subsection, the announcement strategies described in Subsection 4.4.3 are evaluated using the network configurations described below in Subsection 4.4.5. Based on the quality properties of Subsection 4.4.3, the strategies are compared to each other.

To test the strategies, the centralized network configuration was evaluated with different node counts. For each of the eight announcement strategies, the tests were performed using 2, 5, 10, 25, 50, 100 and 200 nodes, resulting in 56 configurations. These configurations were each executed using two nodes starting mechanisms: a) the batch node start, in which all nodes were started randomly in the observation interval window; b) the delayed node start, where a node was added every second, resulting in a linearly growing network.

In addition, two dynamic network configurations were used: *Split*, where the central network was split in two halves, and *Merge*, where two equally sized networks were joined. Summing up the different configurations, 224 independent experiments were performed.

Evaluation Setup

To evaluate the announcement strategies, several setups were used, including emulations with many nodes as well as physical machines connected over various network links.

Network Emulation

For network emulation, the Common Open Research Emulator⁴¹ (*CORE*) was chosen, which is scriptable using *Python* and in this way allows versatile creation of experimental configurations. This system uses Linux and lightweight virtualization to provide a networking testbed for unmodified, regular Linux binaries. All announcement strategies are evaluated under four different network scenarios described below:

Centralized Network In the *Centralized Network* configuration, all nodes are connected centrally and hence are located in the same collision domain. This setup is similar to a classic network hub or a local ad hoc wireless network in the sense that each node can directly communicate with all of its adjacent peers. As long as the network is not oversaturated, every node gets the announcements of every other node.

Growing Network In the *Growing Network* configuration, nodes are added periodically to the network. Ideally, the announcement strategies should adapt to the new situation fast and down-regulate their announcement counts. Each second, a new node joins the network, and adaptation is required to maintain optimal resource usage.

Merging Network In the *Merging Network* configuration, two equally sized *Central Networks* merge at a fixed point in time, doubling their size instantaneously. Using this configuration, adaptation rates for abruptly changing network configurations can be observed.

Splitting Network In the *Splitting Network* configuration, the network is split in two halves at a fixed point in time. By creating two independent networks, the announcement strategies need to react fast to satisfy the defined quality properties and avoid prolonged periods of silence between announcements.

⁴¹<http://www.nrl.navy.mil/itd/ncs/products/core>

4 Disruption-tolerant Device-to-Device Emergency Communication

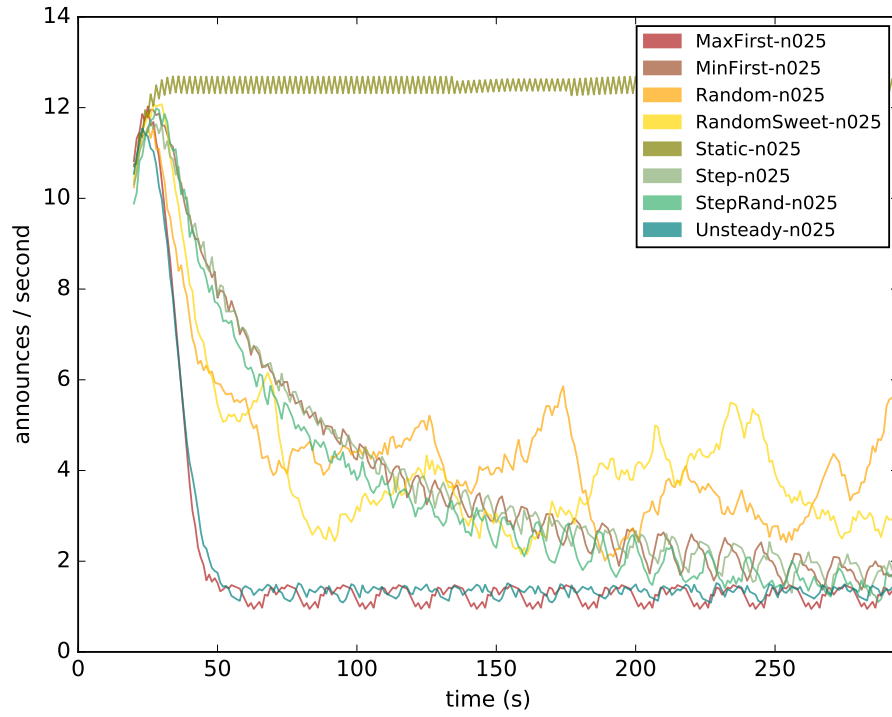


Figure 4.21: Announcements/second in a static network of 25 nodes.

Physical Testbed

To evaluate the proposed announcement strategies under realistic conditions, a physical testbed was created. It consists of several Raspberry Pi 3 Model B⁴² single-board computers, running under the vendor-provided Debian-Linux-based Raspbian⁴³ operating system. This platform is comparable to mobile phones in terms of energy consumption and therefore allows one to obtain realistic energy and power consumption measurements when evaluating the announcement strategies.

Eight Raspberry Pis were setup up as network participants, as well as an additional Raspberry Pi as a system under test (SUT). The energy consumption of the SUT was measured using an *Odroid Smart Power* measurement device, an external power meter. The data points were logged at 5 Hz to another device, in order to prevent disruption of the measurement.

Basic Capabilities

In Fig. 4.21, the announcement rate for all strategies in a static network with 25 nodes is visualized. The strategies share the same observation interval, and therefore the first 20 seconds are the same, since they also start with the same announcement interval of 2 seconds. The *Static* strategy preserves this announcement interval, and the globally generated traffic remains the same for the whole experiment.

Unsteady and *MaxFirst* show very low announcement rates in this network configuration. *Unsteady* uses the node count (see Sec. 4.4.3) and computes its maximum

⁴²<https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>

⁴³<https://www.raspbian.org>

4.4 Optimizing Epidemic Announcements

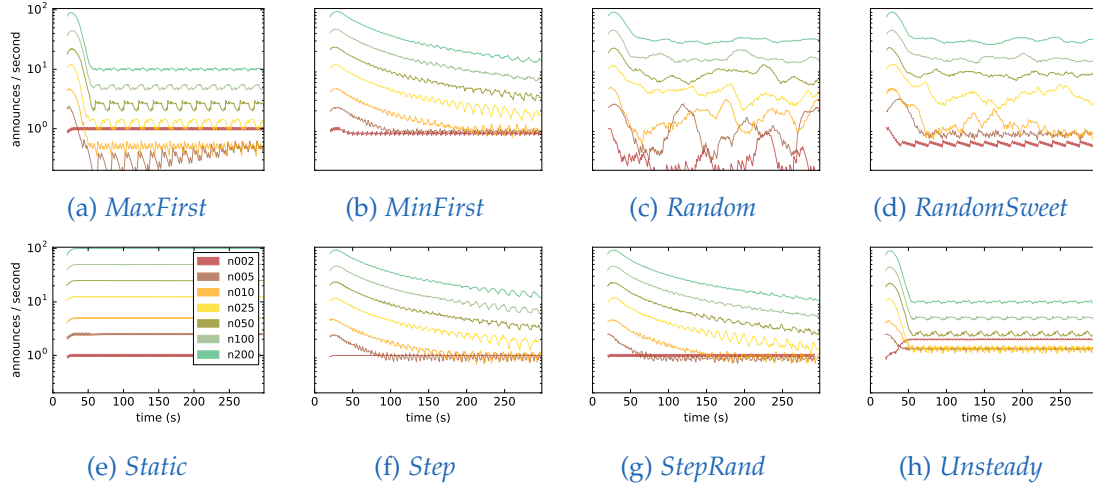


Figure 4.22: Comparison: announcements produced by the proposed strategies in different static network configurations.

announcement delay, which in this case is greater than the observation delay and sets this maximum. Because the observed announcement count is high, *MaxFirst* jumps to the maximum possible announcement delay. Since the situation does not change, both algorithms stick to their decision in future observations. This similarity changes for lower node counts. Considering Figure 4.22a, *MaxFirst* sets the same very low announcement rates in the beginning, which leads to low global announcement rates and finally to big gaps between each two announcements. *Unsteady* (Fig. 4.22h) compensates this problem and starts with higher announcement rates in smaller networks.

MinFirst and *Step* also behave similarly, since the down steps are implemented the same way. Both algorithms extend their announcement delay by 1 second, starting at a delay of 2 seconds. *StepAndRand* also is in the same group and only differs from *Step* by adding a random value with a maximum of 0.5 seconds. All three algorithms achieve the goal of a less saturated network and also approach the same minimum as *MaxFirst* and *Unsteady*.

In this network configuration, *RandomSweet* as well as *Random* show a similar behavior. The announcement rate drops directly after the initial observation, but stays higher than for the other strategies that achieve a low announce rate after around 200 seconds. To get similar results as, for example, *MaxFirst*, all nodes would need to pick a pretty high delay by chance, and the more nodes in the network, the more unlikely it is that all nodes do this in the same observation interval.

Bandwidth Savings

A major goal for using dynamic announce intervals is the reduction of bandwidth in such protocols. Table 4.4 shows the announcement rates of the proposed strategies compared to the static announcement strategy. For this table, the announcements sent by one node in the batch node start is used. This number also includes the observation delay in which all strategies follow the static behavior.

All non-static strategies converge for growing node counts. *Step*, *StepRand* and *MinFirst* use around a third of the number of announcements compared to *Static*.

4 Disruption-tolerant Device-to-Device Emergency Communication

Table 4.4: Announcements of the strategies compared.

# Nodes Name	2	5	10	25	50
Static	291	732	1460	3658	7296
Random	34,4%	47,0%	37,0%	37,9%	37,3%
RandSweet	58,1%	41,7%	29,0%	35,6%	37,7%
Step	101,7%	45,4%	35,2%	33,2%	33,4%
StepRand	99,7%	42,5%	32,5%	30,1%	30,2%
MaxFirst	99,0%	21,2%	17,1%	17,0%	17,1%
MinFirst	84,9%	44,3%	34,7%	33,3%	33,5%
Unsteady	188,7%	56,8%	32,5%	17,7%	17,1%

MaxFirst and *Unsteady* take advantage of their fast adaptation rate and are able to save around 80% of the announcements. This means that only one fifth of the bandwidth is used without sacrificing any comfort or usability of the protocol.

Table 4.4 also shows that the proposed strategies benefit the most from their dynamic behavior for networks with 2 to 10 nodes. After that, only minor improvements can be achieved. The announcement rate of *Static* can be altered easily by hand and could therefore also reach the goal of a lower global announcement rate for big networks, but would then lose the ability to perform good in small networks without manual interaction on each node.

Unsteady uses more bandwidth than *Static* for a minimal network. This allows fast discovery of new peers in an existing network and addresses the real-world problems described in Figure 4.20. *Random* and *RandomSweet* have a lower total announcement count in small networks. This shows that these strategies are inferior in terms of discovery times. The remaining *Step*-based strategies show satisfactory results in small and bigger networks in terms of bandwidth usage, but take a longer time to reach an optimal resource usage.

Adaptation Rate

Unsteady and *MaxFirst* have a very high adaptation rate, since they set their final announcement delay after the first observation interval for all static network configurations, as presented in Figure 4.22h. *MaxFirst* is able to achieve fast adaptation rates for big networks, while *MinFirst* is able to achieve this in small networks, as a result of their designs. A disadvantage of *MaxFirst* is shown in Figure 4.22a: For small networks, the announcement rate also drops to the minimum in the first place, so discovery may be worsened.

The adaptation rate of the *Step*-based algorithms depend on the number of nodes. As outlined in Figure 4.22f, in a network of 5 nodes around 70 seconds and in a network of 10 nodes around 150 seconds are needed to fully adapt.

In Figure 4.23, a splitting network configuration with 10 nodes is presented. The *Step*-based strategies reach their target announcement rate immediately. In *RandomSweet* and *Unsteady*, new announcement rates are visible after about 30 seconds. Both strategies reach announcement rates as in the united, central network. This understanding only slightly differs in the merging network: The *Step* based algorithms need longer, while

4.4 Optimizing Epidemic Announcements

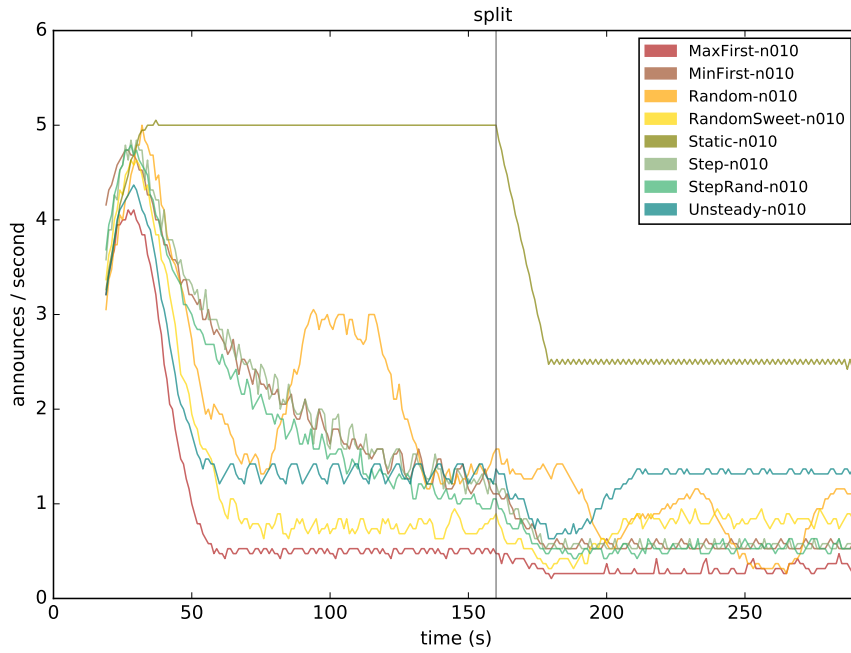


Figure 4.23: Splitting network configuration with 10 nodes.

Unsteady and *MaxFirst* adapt in the observation interval.

The observed adaptation rates are also valid for the merging network configuration: *MaxFirst* and *Unsteady* adapt in a 30-seconds window, while the *Step* strategies take a longer time. For the network of 5 nodes, the *Step* strategies also achieve an adaptation rate of around 40 seconds. Especially in small networks, this rate is important, since the announcement gaps are compensated quickly.

Figure 4.24 shows a delayed start of 100 nodes, with one node starting per second. Compared to *Static*, the proposed algorithms are able to keep the announcement rates low. Since every node announces using the default interval for the first 20 seconds, the announcement rate grows even in the very agile *Unsteady* and *MaxFirst* strategies. Immediately after all nodes are spawned, the algorithms are able to adapt to the situation.

Announcement Gaps

Figure 4.25 shows a violin plot of the global announcement gaps for a static network with 10 nodes. The mean gap correlates with the global announcement rate, and so does the variance. Having this in mind, the perceptions of Subsection 4.4.5 are backed by this plot. Although *MaxFirst* does not have the highest announcement gap, it produces a relatively high percentage of longer gaps, while all other strategies only have a low number of outliers in this area. This is also the case for a network of 5 nodes. Yet larger network configurations do not show the same characteristics. This behavior can be ascribed to the observations made in the previous section.

What stands out is that compared to *Static*, all strategies perform worse with respect to the maximum announcement gap. This can be put in perspective by examining the upper quartile: For all algorithms except for *MaxFirst*, the upper quartiles of the

4 Disruption-tolerant Device-to-Device Emergency Communication

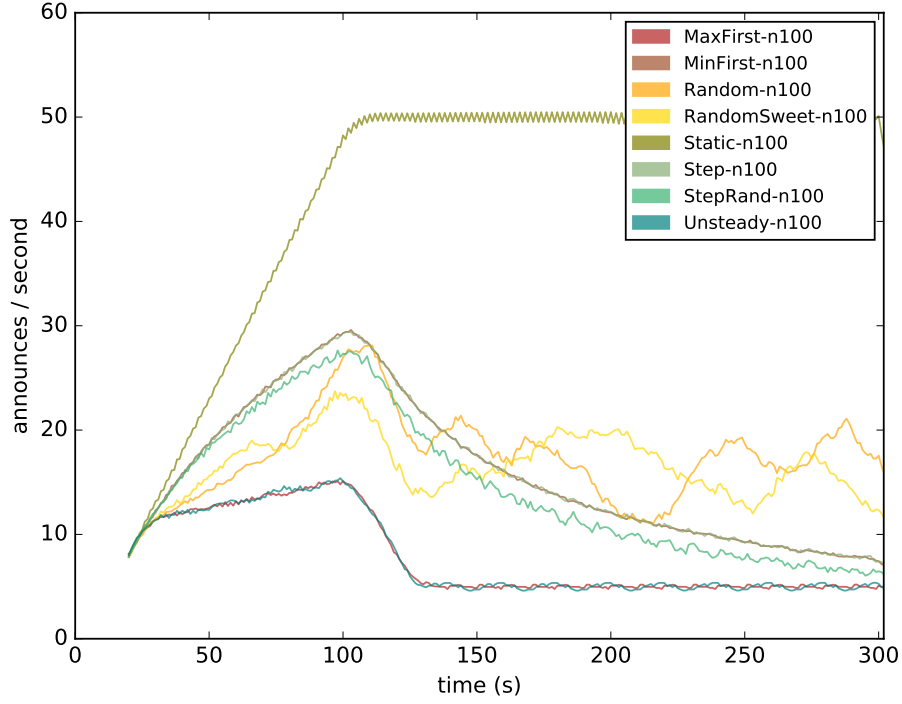


Figure 4.24: A growing network with 100 nodes.

announcement gaps are below 2 seconds.

Energy Consumption

The initial assumption was that a reduced number of announcements would reduce the consumed energy proportionally. This assumption was evaluated in a wireless network of 9 ARM-based nodes as described in Section 4.4.5. In these experiments, each node acted as sender and receiver simultaneously. One node (system under test - SUT) was connected to an external power meter (ODROID SmartPower), which logged the power and energy consumption of the node at a 5 Hz rate. Additionally, every experiment was performed with two different network interface configurations, with a different idle power consumption each: ad hoc mode ($P_{idle}=1.37$ W) and managed mode ($P_{idle}=1.45$ W).

To measure the higher end of the power consumption, two additional announcement strategies sending announcements at a high rate are introduced: *Static05* and *Static01*, with 2 and 10 announcements per second, respectively.

To compute the energy consumed by the software, the average idle power is subtracted from the measured power in the given 300 seconds measurement interval:

$$E := \int_0^{300} P_{measured}(t) dt - 300 * P_{idle} \quad (4.1)$$

In the physical testbed with 9 nodes, the default *Static* strategy uses 1.99 mWh. *Static05* and *Static01* use 11.97 mWh and 32.52 mWh for their announcements, respectively. Based on these numbers, a correlation between the number of announcements (sent and received) and the consumed energy is found and presented in Table 4.5.

4.4 Optimizing Epidemic Announcements

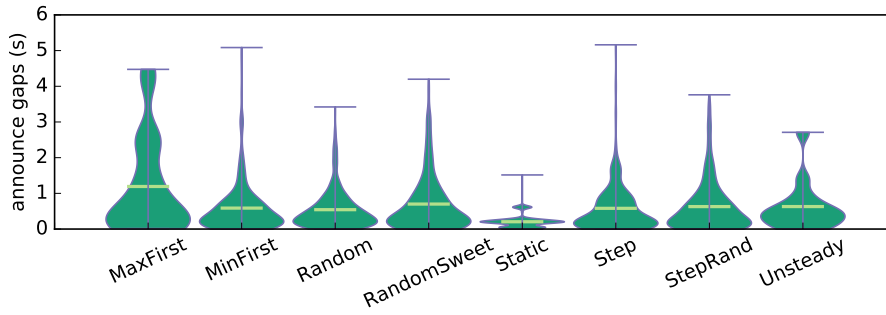


Figure 4.25: *Announcement Gaps* in a static network of 10 nodes.

Table 4.5: Correlation of energy consumption and announcements in a physical testbed of 9 nodes.

Name	# Ann.	E (mWh)	rel. Ann.	rel. E	ratio
Static	1323	1.99	1.00	1.00	1.00
Static05	5404	11.97	4.08	6.00	1.47
Static01	29342	32.52	22.18	16.31	0.74
MaxFirst	256	1.17	0.19	0.59	3.04
MinFirst	473	1.26	0.36	0.63	3.04
Random	434	1.34	0.33	0.67	2.04
RandomSweet	342	0.73	0.26	0.37	1.42
Step	495	1.20	0.37	0.60	1.61
StepRand	460	1.12	0.35	0.56	1.61
Unsteady	514	1.38	0.39	0.69	1.78

While the correlation between the number of announcements and the energy consumption is reasonable for large numbers of announcements, this correlation is not substantial for lower numbers of announcements. The general trend seems to be correct (correlation coefficient $r = 0.985$), since all proposed strategies need less energy than *Static*. In contrast, there are examples in which this correlation seems to be vice versa, e.g., when comparing *MaxFirst* and *RandomSweet*.

To summarize, the energy measurements of the experiments show that for high numbers of announcements the energy consumption is increased. Side-effects of the programming language, as well as the relatively low energy impact of the announcements of *Mesher* disturb the energy measurements. Nevertheless, a general trend is clearly evident.

4.4.6 Conclusion

In this section, it was shown that without relying on application-specific properties, optimizations for network protocols relying on announcements can be achieved. Eight different announcement strategies were compared, including a standard static announcement strategy and a random announcement strategy. While a random announcement strategy might preserve more bandwidth than a static announcement strategy, it has negative side-effects compared to the other proposed announcement strategies. By

4 Disruption-tolerant Device-to-Device Emergency Communication

dynamically changing the announcement interval and depending on the number of nodes involved, the bandwidth required for announcements could be reduced by more than 80% compared to a static announcement strategy. Nevertheless, the requirement of fast discovery of at least one node is still met. The evaluation of the proposed announcement strategies in terms of energy consumption show that announcements do effect battery lifetimes and are thus worth to be reduced.

There are several areas for future work. For example, so far the algorithms only have access to information like the number of announcements received in the last observation interval or the number of currently known peers. By giving the strategies more information, further optimizations might be possible. Furthermore, a dynamic observation interval could be implemented, to allow even faster adaptation to new situations.

4.5 DTN-RPC - Offloading Work in Challenged Environments

4.5.1 Introduction

The possibility of calling a procedure on a remote computer has been introduced to program client-server interactions in a procedural manner. *Remote Procedure Calls* (RPCs) [78] have proven to be useful in many distributed computing scenarios to simplify application programming by eliminating the need for explicitly having to code the details of remote interactions based on a request-response message-passing protocol. RPCs have been integrated into programming languages (e.g., Java RMI, Python RPyC, Distributed Ruby DRb-RPC, and Erlang RPC), dedicated applications (e.g., SAP RFC), and WWW protocols (e.g., XML-RPC, JSON-RPC, SOAP, Windows WCF, Google gRPC, Google Web Toolkit RPC).

However, none of the existing RPC implementations are designed to work properly for Delay/Disruption-Tolerant Networking (DTN) [79], [80] where network connectivity is periodic, intermittent, prone to disruptions, and a direct connection to a remote server might not exist. DTN scenarios with potentially large transmission delays as a result of either inadequate physical link properties or extended periods of network partitioning are common in natural disasters. For example, during the 2010 earthquake in Haiti, public and mobile telephone systems were destroyed or disturbed and could not be rebuilt or repaired for days⁴⁴. An inoperative cellular communication infrastructure during the earthquake in New Zealand on November 14, 2016, created uncertainty about whether people were still in the affected areas⁴⁵. Even in the absence of disasters, there are still regions, e.g., in India [81] and Australia [82], where no telecommunication infrastructure exists and where people cannot communicate using mobile devices. Whenever reliable end-to-end connectivity is not available, DTN can be used to sustain communications without requiring any conventional infrastructure.

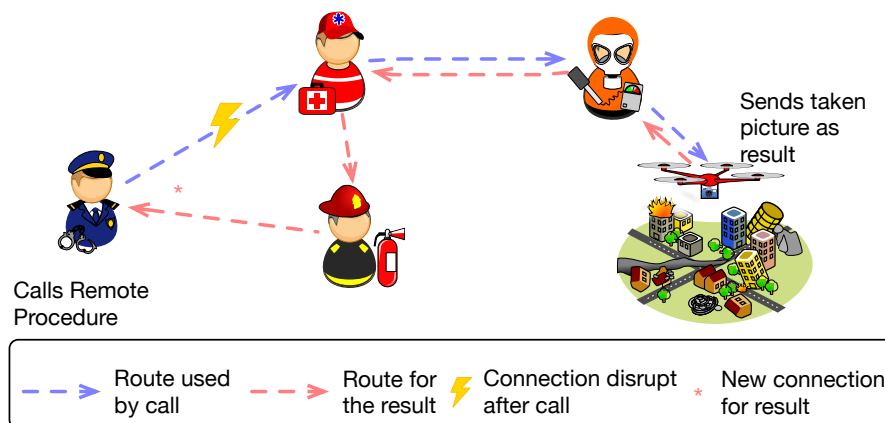


Figure 4.26: Calling a remote procedure in a DTN disaster scenario.

Being able to use RPCs in these scenarios could provide great services for civilians and professional first responders. For example, quadcopters could offer a procedure that takes a picture with a mounted camera at a particular geographical location and

⁴⁴https://en.wikipedia.org/wiki/2010_Haiti_earthquake

⁴⁵<https://www.bbc.com/news/world-asia-37970775>

returns it over the network. Then, rescuers could request an overview image via an RPC to a quadcopter while performing other tasks until the file arrives over a DTN connection using nodes of other rescue workers or citizens as relay nodes. This example is illustrated in Fig. 4.26, where the call takes the blue route, but the result arrives over the red route due to the connection loss illustrated by the yellow lightning symbol. This might take longer, but without DTN the call could not be made at all.

In this section, *DTN-RPC* is presented, a new approach to provide RPCs for DTN environments. *DTN-RPC* relies on (a) control and data channels to cope with potentially short contact durations in DTN where it is impossible to transmit large amounts of data, (b) explicit and implicit modes for server addressing, (c) Non-DTN and DTN transport protocols for calls and results, and (d) predicates that servers can check to decide whether a procedure should be executed. The open-source implementation of *DTN-RPC*⁴⁶ is based on Serval [2], [27]–[29], an open-source, disruption-tolerant wireless ad-hoc networking system. The experimental results obtained within the network emulation framework CORE indicate that the measured CPU and network overheads for *DTN-RPC* are reasonably low so that *DTN-RPC* can be executed on smartphones or routers, and that the round-trip times and the number of successful RPCs are highly satisfactory in dynamically changing network topologies with unreliable connectivity.

Parts of this section have been published in [4].

4.5.2 Related Work

Tu and Stewart [83] present a Java RPC framework where small data is replicated and sent over a second TCP connection to the server or back to the client. At the destination, a listener collects all arriving data on all connections, reassembles the original data, and passes it to the corresponding handler.

Stuedi et al. [84] increase the efficiency of RPCs in data centers by softening the userland and kernel separation in the network stack and by using remote direct memory access to minimize the overhead of network operations by performing them with less context switches and zero-copy network I/O.

Chen et al. [85] introduce memory regions where server and client exchange data to improve the efficiency of RPCs between virtual machines (VMs) on the same host computer. The proposed framework has three components: (a) a notification channel that informs the server about new calls and the client about arriving results, (b) a control channel that sends meta-data (e.g., the parameter count), and (c) a transfer channel that is responsible for transmitting data between server and client and putting the data in the predefined memory regions.

Shyam et al. [86] propose solutions for situations where an RPC server is not available. The first solution is a heartbeat server that observes whether the RPC server is operative. The second solution is that every node sends a health check message to the RPC server. Since these messages are typically smaller than an RPC request and no computations take place, the answer of the health check should arrive faster. If the answer does not arrive within a timeout that is smaller than the timeout for the RPC, the server is considered inoperative.

Reinhardt et al. [87] address the problem of providing RPCs in wireless sensor networks. In particular, the authors eliminate the need of conventional RPCs to send

⁴⁶<https://github.com/umr-ds/DTN-RPC>

4.5 DTN-RPC - Offloading Work in Challenged Environments

predefined data to predefined destinations, typically addressed by ports, by publishing descriptions of new sensors that can be used by other sensors or nodes dynamically.

Shi et al. [88] present a framework where mobile devices can offload jobs to other mobile devices. In scenarios where node mobility is high, only small tasks will be offloaded; otherwise larger jobs will be offloaded, too. To increase the number of offloaded jobs, every job is split into smaller tasks. Additionally, every node has to announce its capabilities, such as CPU capacity and available battery power. To offload a job, the framework compares the task requirements with the capabilities of the client and tries to find a server that satisfies the requirements better than the client. If no server is found, the job will be executed locally.

Chen et al. [89] propose a solution for offloading computations to ad-hoc cloudlets. A job is offloaded via an ad-hoc communication channel that is closed after the procedure has been called successfully. The result of the job can arrive (a) via an ad-hoc channel if server and client are in close proximity, (b) via a cellular network used when an ad-hoc connection is not possible, (c) via a WiFi access point, if available.

Zhang et al. [90] propose a solution for cloudlets with intermittent connectivity where parts of a job will be executed either locally or remotely. The decision which of both options is chosen is based on a probability that includes the cost of executing a task. Two cost factors are calculated: (a) the cost when the phase is executed locally, where, e.g., energy consumption is important, (b) the cost when the phase is executed in a cloudlet, where, e.g., available bandwidth is important. Based on this information, a Markov chain can be constructed and the optimal path can be found.

Lai et al. [91] propose an offloading algorithm for delay-tolerant mobile networks that increases the amount of offloaded data without increasing the transmission overhead or delay. The transfer channel is chosen based on the contact duration between two nodes and the available transmission protocols. Therefore, every node logs which neighbors are available. Based on the available neighbors, on the size of the data that is offloaded, and the estimated waiting time, a priority is computed. With these factors, a utility is calculated that denotes whether data should be offloaded using this particular channel or not.

To summarize, several of the related works address problems of RPCs in traditional networks, where links are either static or tasks are on the same machine, such as in VMs. Furthermore, direct memory access methods to reduce networking overhead cannot be used in a DTN environment, due to possibly untrustworthy nodes. Also, control mechanisms like heartbeats or duplicating data on multiple channels are no options for DTN. In the offloading approaches, the particular problems of RPCs in DTN are either not addressed or would require additional infrastructure, such as cell towers for 3G or LTE connectivity, or nodes with access to the Internet. *DTN-RPC* on the other hand is designed to provide RPCs in DTN environments without requiring any additional infrastructure.

4.5.3 DTN-RPC's Design

This section presents the design of *DTN-RPC*.

Fundamental Considerations

There are several differences between RPCs in traditional networks and RPCs in DTN.

4 Disruption-tolerant Device-to-Device Emergency Communication

In conventional RPC implementations, errors are handled, for example, if the connection between client and server is lost. In DTN, it is not certain whether a call even reaches its destination. Thus, errors in DTN can only be handled in a few situations, since error reports could just not arrive and the client would not notice that the call was not successful. The server, on the other hand, would have to spend computational overhead while trying to inform the client about the error. Furthermore, disruptions and poor connection quality make it impossible to support real-time communication or to guarantee a predefined quality of service in DTN.

Common RPCs are location transparent. For this purpose, stubs or proxy functions exist to handle communication via the network. In DTN, a call will explicitly be executed remotely, and it is expected that there will be networking overhead when executing a remote procedure.

In several RPC implementations, the client has to register at the server before calling a procedure. Since in DTN the address of a server is typically not known, client registration is not possible.

Traditional RPC servers either announce the procedures they offer or there exists a lookup service where clients can find information about which server offers which procedure. In DTN, server announcements might not reach or lookup services might not be available for clients when needed.

Control and Data Channels

DTN is often used in mobile mesh and ad-hoc networks where the network topology changes frequently. This can lead to short contact durations between nodes where it is impossible to transmit large amounts of data. Due to this restriction, two separate communication channels are introduced in *DTN-RPC*: the *control* and the *data* channel.

The *control* channel is responsible for transmitting meta-data, such as the procedure name and the parameters, from client to server, and possible results from server to client. The *control* channel supports two modes to address remote servers, *explicit* and *implicit* (*any* or *all*), as described below.

Explicit If the address of a server is known and the server is reachable, *DTN-RPC* will choose the *explicit* mode and will try to establish an end-to-end connection to this specific server.

Implicit (*any* or *all*) If the address of a server is not known, but potential servers are reachable, both the *any* and *all* modes (summarized as the *implicit* mode) are used to broadcast a call. In the *any* mode, the client waits for exactly one response. This is helpful if it is known that servers exist that offer a particular procedure, but it does not matter which server responds. The first arriving response will be accepted. In the *all* mode, the client will wait for as many answers as possible until its internal timeout occurs. This is useful in scenarios where the quality of the results varies with the executing machine (e.g., GPU support, different algorithms), where different answers should be combined (e.g., to implement aggregate functions that return a value across all items in the results set), or is influenced by other factors such as geolocation (e.g., sensor readings, taking a picture).

4.5 DTN-RPC - Offloading Work in Challenged Environments

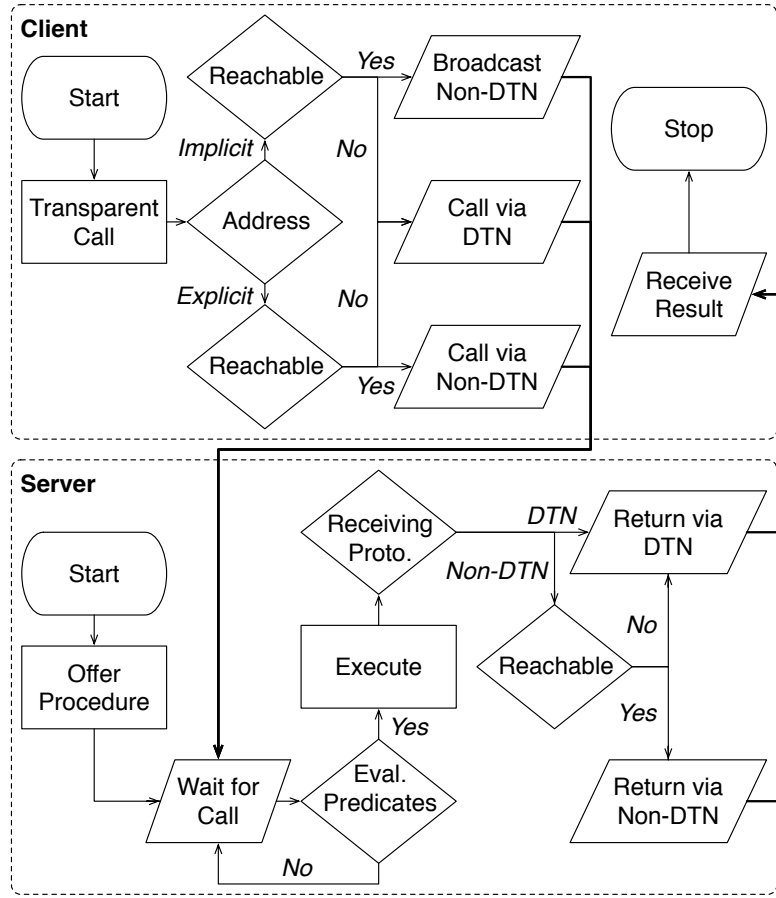


Figure 4.27: DTN-RPC flowchart for client and server.

The payload of the *control* channel packets must not exceed the payload size of the underlying transport protocol to keep the data on the network as small as possible.

The *data* channel transports larger amounts of data from client to server and vice versa. It is used if a file is required as a parameter for a particular call. The transport of the payload in the *data* channel is always performed via DTN. The transport of the meta-data in the *control* channel is explained below.

Transparency

In both *explicit* and *implicit* addressing modes, the *control* channel of DTN-RPC supports *Non-DTN* and *DTN* transport protocols and automatically switches between them for performing a procedure call, as explained below.

Non-DTN vs. DTN As illustrated in Fig. 4.27, if the server is reachable in the *explicit* mode, DTN-RPC will use a Non-DTN transport protocol to call the server. If the server is not reachable, the call will be issued using a DTN protocol.

After having called a remote procedure in the *explicit* mode, the client waits for the response using the same transport protocol that was used to call the procedure. If the connection is interrupted, the client additionally waits for results that arrive via a DTN protocol.

4 Disruption-tolerant Device-to-Device Emergency Communication

After having successfully executed a received call, the server checks whether the explicit control channel on which the call was received via a Non-DTN protocol is still available, as shown in Fig. 4.27. If the channel is not available anymore, the result will be sent via a DTN protocol. The *DTN-RPC* server does not attempt to re-establish a Non-DTN connection, since it is unlikely that a reconnection is successful if one of the nodes has physically moved out of the network's reach. If the call was received via a DTN protocol, the server also uses a DTN protocol for its response.

Since the *implicit* modes use broadcast addresses to call procedures, a different transport protocol has to be used than in the *explicit* mode, because reliable point-to-point transport protocols like TCP do not support broadcast packets. Since a server availability check in a broadcast scenario would imply communication between multiple nodes, which would add additional delays, a call just gets broadcasted without any prior availability checks. If a timeout occurs and no result arrives, the call is performed via a DTN protocol.

Transparent *DTN-RPC* is designed to automatically select the most suitable transport protocol in any given scenario. In the *transparent* transport method, both client and server are designed to make all the above discussed decisions without any user interaction.

Offering and Executing Calls

To offer a remote procedure as shown in Fig. 4.27, two steps are required on the server: declaring and implementing a procedure. The first step of offering a remote procedure is that every procedure has to be declared as a prototype in an extra configuration file in order to tell the server which procedures are available for execution. The implementation of a procedure, which is the second step, has to be provided as an external executable written in any programming language.

The parameters of an incoming call are passed in the order they were received to the external program that then executes the procedure. After the procedure finishes, the result is returned to the server that marshals the result and prepares the result to send it back to the client.

Typically, the computational resources and the battery lifetimes of nodes in DTN are limited. To avoid the execution of calls that would consume too many resources with respect to the current state of a server, a server can decide whether a remote procedure should be accepted or not. For this purpose, particular predicates can be defined, such as thresholds for resource constraints (number of concurrent processes, remaining battery life etc.) or available (sensor) hardware like GPS. This is also shown in Fig. 4.27. The server checks whether defined predicates are satisfied. If at least one requirement is not met, the procedure will not be executed.

Furthermore, each call can provide its own requirements that also have to be checked by the server. For example, some calls should only be executed on non-moving nodes, or require special sensor hardware or extensive resources, such as disk space or RAM. Therefore, there is a two-stage predicate check per server: the first one is the general server acceptance check, and the second one is call-specific and evaluated after having passed the first check.

4.5.4 Implementation

The implementation of *DTN-RPC* is based on the Serval Project [27]–[29]. Serval is centered around a suite of protocols designed to allow ad-hoc and infrastructure-independent communications. The Serval Mesh Protocols abstract from lower-layer protocols, such as IP, UDP, WiFi, packet radio or others. Serval’s real-time packet-switched protocol is the Mesh Datagram Protocol (MDP), which can be compared to UDP/IP, but uses SIDs (Subscriber ID, the public key of an asymmetric elliptic curve key pair) instead of IP addresses, and includes encryption, authentication and integrity features by default. To route packets, MDP uses a protocol inspired by OLSR⁴⁷ and B.A.T.M.A.N. [30] for both node discovery and maintaining a routing table, which facilitates multi-hop routing of packets. On top of MDP, the Mesh Streaming Protocol (MSP) provides reliable data streaming, similar to TCP. Finally, Rhizome is a simple store-and-forward protocol defining files as *bundles*. Intended as the DTN protocol of Serval, Rhizome uses an epidemic routing protocol to transmit files hop-by-hop from source to destination. Rhizome is purposely agnostic of the transport protocols below it, requires no routing table and focuses on single-hop communications, with multi-hop communications emerging as a natural consequence of *bundles* replicating among nodes. *DTN-RPC* uses MDP, MSP, and Rhizome to handle different situations and addressing modes.

As shown by the evaluation in Section 4.3 Serval is an elaborate and ready-to-use software for DTN and mesh networks.

For programmers, an API is offered that can be used to develop programs using the *DTN-RPC* library to execute procedures on remote devices in DTN environments.

Calling a Remote Procedure Transparently

To call a remote procedure transparently, a single function is required that is part of the offered *DTN-RPC* API. This function has five parameters: the server address, the name of the called remote procedure, the number of parameters of the procedure, the parameters themselves and the execution requirements discussed in Section 4.5.3. The mode to be used is determined by the first parameter of this API function call.

Explicit If the parameter is a valid address, the remote procedure will be called explicitly, i.e., the call will be issued via Serval’s MSP, if the server is available. A routing table is built in an ad-hoc manner. If the address of the server can be found in this routing table, this particular server is reachable. While waiting for the result, the client checks periodically whether the connection is still alive. If the connection terminates, the client starts a Rhizome DTN listener.

Implicit The modes *any* and *all* are used if the address is the *ANY* address provided by Serval for *any* or the broadcast address for *all*. Since Serval’s MSP supports point-to-point communication only, it is not possible to send data to the broadcast address. Therefore, *any* and *all* use Serval’s MDP.

Since a reachability test is not possible for broadcast packets, the procedure will be called without any prior checks. Since delivery is uncertain, the client sends a call every

⁴⁷<https://tools.ietf.org/html/rfc3626>

4 Disruption-tolerant Device-to-Device Emergency Communication

second until at least one server responds with an acknowledgement or a timeout occurs. If an acknowledgment arrives, the threshold for the timeout is increased. Only if the new timeout occurs, the client will additionally start a Rhizome DTN listener and wait for the result via DTN.

The difference between the modes *any* and *all* is the number of results. In the first case, the client stops listening as soon as the first result arrives. In the second case, the client waits for as many results as possible, but at least for one.

Returning the Result Transparently

While executing the called procedure, the server does not check periodically whether the client is still reachable. Instead, this check is done once when the response is ready to be sent. If the call arrived via MSP or MDP, but the connection is broken or the client is not reachable, sending will fail and the server will send the result via Rhizome.

4.5.5 Experimental Evaluation

In this section, an experimental evaluation of *DTN-RPC* for different network topologies and in various configurations is presented. Due to the lack of comparable RPC implementations that can handle disruptive networks, *DTN-RPC* is not compared against other approaches. A comparison with widespread software solutions such as JSON-RPC or SOAP would be unfair, since they would fail each time the network connection is lost.

Test Setup

The evaluation of *DTN-RPC* is based on the open source network emulation framework CORE⁴⁸. Compared to protocol simulations, CORE can run *DTN-RPC* without modifications in a more realistic Linux environment. All tests are performed on a 64-core AMD Opteron 6376 CPU with 256 Gigabyte RAM, emulating up to 64 virtual nodes at the same time.

Measurements Standard Unix tools are used to measure system properties with a time resolution of one second. For CPU statistics, *pidstat*⁴⁹ is used, and the Serval and *DTN-RPC* processes are monitored from within a node. Network usage is measured from within the nodes on every network interface for Serval and *DTN-RPC* using a custom Python script based on *libpcap*⁵⁰. To monitor the behavior of *DTN-RPC*, metrics such as call times, round-trip times, and logging functions were implemented and integrated into the binary.

Network Topologies Three network topologies are considered, as shown in Table 4.6.

⁴⁸<https://www.nrl.navy.mil/itd/ncs/products/core>

⁴⁹<http://sebastien.godard.pagesperso-orange.fr>

⁵⁰<http://www.tcpdump.org>

4.5 DTN-RPC - Offloading Work in Challenged Environments

Table 4.6: Topologies

Name	# Nodes	Description
<i>Hub</i>	28	All nodes connected to each other
<i>Chained</i>	32	Pair-wise connected
<i>Islands</i>	64	Partitioned islands with dynamic links in between

Hub The *Hub* topology connects 28 nodes with each other so that every node is one hop away from all other nodes. As shown previously in Section 4.3, the Hub topology is challenging for Serval and thus also for *DTN-RPC* due to the high number of direct neighbors, all using bandwidth and flooding each other with status information. Therefore, the *Hub* topology helps to investigate whether *DTN-RPC* can handle RPCs when the network is under heavy load.

Chained The *Chained* topology consists of a chain of 32 nodes, 31 hops from the first to the last node. Typically, network connections over the Internet require less than 16 hops. In a DTN mesh network, more hops might be needed for messages to reach their destination.

Islands The *Islands* topology represents a partitioned, dynamic network with 64 nodes. At the beginning, there are 4 islands each containing 16 nodes. The 16 nodes per island are connected randomly with each other, creating an ad-hoc mesh network. Then, four different behaviors can occur randomly every 60 seconds: two islands are connected, two connected islands are disconnected, all islands are connected or all islands are disconnected resulting in the original state.

Network Connections *DTN-RPC* adds a new layer of abstraction to the Serval networking stack. Although Serval can cope with several degraded networking scenarios, *DTN-RPC* is only evaluated in situations where network connections are completely lost, because this is the most challenging situation in DTN. Network degradations and bandwidth limitations would only lead to higher delays, but not break *DTN-RPC* itself.

Test Sets and Modes The remote procedure used in the tests implements a simple echo service. It is called with three different test sets: (a) *0MB*, where no file is used; (b) *1MB*, where a file of 1 megabyte is transmitted; (c) *100MB*, where a file of 100 megabyte is sent.

Additionally, all tests are executed in 10 different modes: *explicit*, *any* and *all* via Rhizome; *explicit*, *any* and *all* via MDP; *explicit*, *any* and *all* transparently and *explicit* via MSP.

Servers Since the successful execution of remote procedures in DTN depends on the number and distribution of servers, every test in *Hub* and *Islands* is executed twice, first with 5% of the nodes as servers and second with 50%. In *Chained*, the goal is to determine how *DTN-RPC* performs if the call has to travel a long distance. Thus, only one server and one client at the opposite ends of the chain are needed.

4 Disruption-tolerant Device-to-Device Emergency Communication

In each test setup, the procedure is called 30 times to get reliable results. The acknowledgement from the server has to arrive within 30 seconds on the explicit channel. After the acknowledgement, the client waits an additional 90 seconds for the result. If within these 90 seconds no results arrived, the procedure is called via DTN, which has an additional 90 seconds to finish. After the client has received the result or all timeouts are reached, the next procedure will be called.

Since our evaluation is concerned with the overhead and the performance of *DTN-RPC*, the possibility of *DTN-RPC* to perform predicate checks to decide whether a remote procedure should be accepted has been disabled in the experiments.

Fundamental Properties

In *Hub* where each node is a single hop away from all other nodes and Serval uses broadcast packets to announce meta-data, each node produces a flood of data that is sent to all neighbors. Thus, both the CPU usage and the network load in *Hub* are always higher than in the corresponding tests in *Chained* or *Islands*, due to the high number of direct neighbors. Furthermore, *DTN-RPC* does not only use the API, but also the networking stack and the communication mechanisms provided by Serval. Thus, *DTN-RPC* cannot be measured separately, but only together with other Serval traffic.

Similar to the network usage, the CPU utilization has to be measured not only for *DTN-RPC*, but also for Serval running on a node. The evaluation of the CPU usage shows that the CPU consumption of *DTN-RPC* is negligible with about 1% in heavy load situations. However, the Serval process has a higher CPU usage, since Rhizome computes a hash for each file sent. The larger the file, the more time-consuming the hash computation becomes. *DTN-RPC*, on the other hand, is independent of file sizes, because it simply issues a call to the Rhizome API, which leads to the described 1% CPU utilization increase in the *DTN-RPC* process. Therefore, since the CPU utilization is dominated by Rhizome, in the experiments below it is always based on the Serval process.

Network Performance

For the *oMB* tests in the *Chained* topology, the overall network load averages at about 2 Mbit/s for each of the three transport protocols (MDP, MSP and Rhizome). This is true for all three modes, *explicit*, *any* and *all*. Since *DTN-RPC* uses only a single packet for calling the remote procedure and returning the result in *oMB*, these packets get lost in the overall network load that is produced by Serval exchanging meta-data and therefore not plotted in Fig. 4.28.

During the *1MB* and *100MB* test sets, the network load increases up to 70 Mbit/s for *1MB* and up to 500 Mbit/s for *100MB*, as indicated by the blue and red graph of Fig. 4.28a, in which the stacked bandwidth for all network interfaces together with the CPU usage in a logarithmic scale for 5 calls with the *100MB* test set and 30 calls with the *1MB* test set is shown. In the *1MB* and *100MB* calls, a file always has to be transmitted via the Rhizome DTN for calling the remote procedure and receiving the result. The difference between the *1MB* and *100MB* calls is due to the different file sizes.

The *Hub* topology shows a similar behavior, as illustrated by Fig. 4.28b, where the stacked bandwidth for all network interfaces together with the CPU usage in a

4.5 DTN-RPC - Offloading Work in Challenged Environments

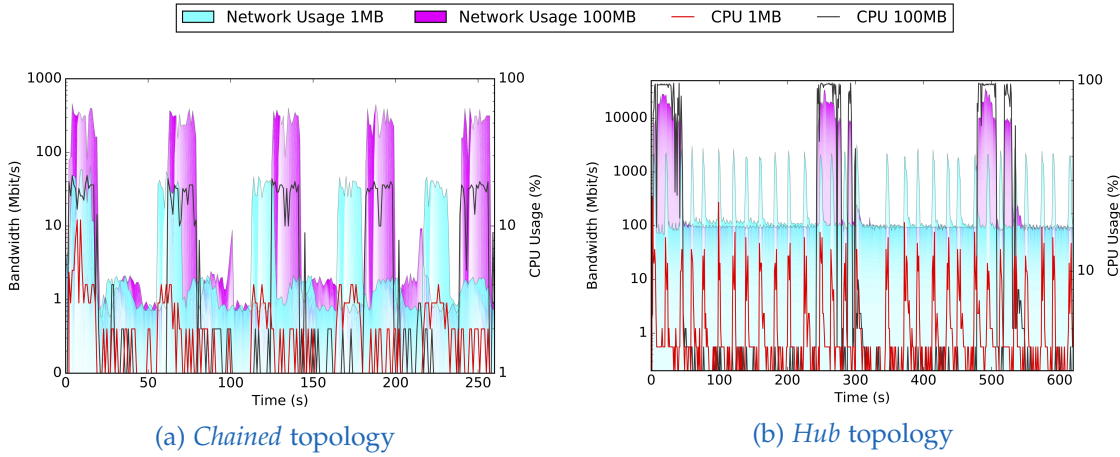


Figure 4.28: Stacked bandwidth usage for 1MB and 100MB and maximum CPU usage for 1MB and 100MB in different topologies.

logarithmic scale for 3 calls with the 100MB test set and 30 calls with the 1MB test set is shown. The main difference is that the *Hub* topology suffers from the problems discussed in Sec. 4.5.5. The overall network usage for the 1MB test sets exceeds 1,000 Mbit/s (blue graph) and 10,000 Mbit/s for the 100MB test sets (red graph).

Comparing the bandwidth consumption to previous results in Section 4.3, DTN-RPC does not add any measurable network traffic to the traffic produced by Serval, and thus can handle scenarios where the network has a high bandwidth usage well.

CPU Usage

As shown in Figures 4.28a and 4.28b, CPU usage highly correlates with network usage. Since CPU usage in the 0MB tests does not exceed 1% after the initial discovery phase, it is not plotted in Figures 4.28a and 4.28b. For the 1MB tests, the maximum is at about 2% up to 3% (red line) and up to 20% for the 100MB tests (black line) in the *Chained* topology.

In the *Hub* topology, the behavior is comparable to the *Chained* topology, with the difference that the CPU usage is generally higher. In the 1MB tests, the CPU usage increases up to about 10% and for the 100MB tests up to 90% during the sending phase. This relatively high CPU consumption happens only while a hash of a file is computed and the file is inserted into the Rhizome store, and thus only during a relatively short time period. As already mentioned, the CPU usage of DTN-RPC does not exceed 1%.

Round Trip Times

To measure the round-trip times (RTTs), only the *Chained* and *Hub* topologies are considered, since the *Islands* topology would not give any credible results due to the random merging and separation of the islands. RTT is only used to indicate the time that is needed to transmit the payload through the network to be sure no additional delays are introduced by DTN-RPC. The execution of a procedure typically takes longer to finish than the implemented echo service.

As shown in Fig. 4.29a, the 0MB tests in *Chained* called by MDP or MSP (i.e., Non-DTN) are executed within a second. As the files grow, the RTT increases.

4 Disruption-tolerant Device-to-Device Emergency Communication

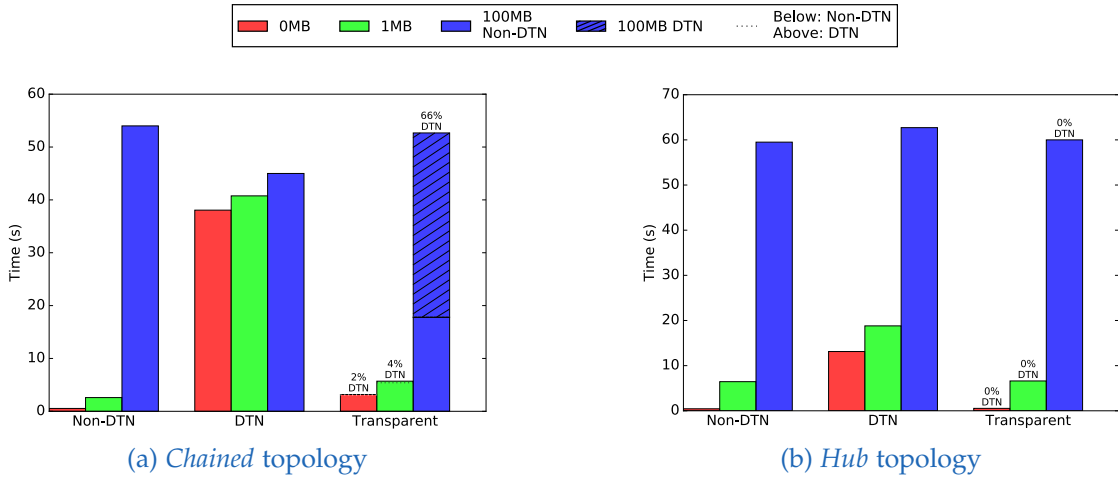


Figure 4.29: Round trip times in different topologies.

In the DTN tests, the RTTs are similar, regardless of the file size. Due to the fact that in DTN the control channel as well as the data channel are transferred via Rhizome, both server and client have to wait for two files. Therefore, all tests take about 40 seconds.

Transparent calls are slower than the calls via MDP or MSP for the *0MB* and *1MB* tests. Some of the calls are issued via MDP or MSP, while others are executed via Rhizome, as explained in Section 4.5.3. The illustrated RTTs are averaged over 30 calls, including the slower Rhizome calls. Furthermore, the time it takes to wait until the transport protocol will be switched is also part of the RTT. Therefore, the *transparent* tests are slower than the corresponding *explicit* tests, but faster than the DTN tests. Since all *100MB* tests are issued using Rhizome and the switch time is included in the RTT, the time it takes for finishing is higher than for MDP or MSP.

As shown in Fig. 4.29b, the RTTs for tests in *Hub* do not differ much from the tests in *Chained*. The only difference is that the *0MB* and *1MB* tests are faster in *Hub*, because all nodes are only one hop away from each other.

To summarize, *DTN-RPC* can execute remote procedures satisfactorily fast. The fallback method using Rhizome is slower, but still can get a result back to the client within an acceptable time, even if the files are large.

Transparency Behavior

In this section, it is examined how *DTN-RPC* behaves in the dynamic *Islands* topology with different numbers of available servers. The figures below show how many of a total of 30 procedures are called using Non-DTN or DTN, respectively, in terms of percentage values. The left half of the pie charts represents outgoing calls and the right half incoming results.

Since the *Islands* topology consists of 4 islands with 16 nodes that merge and separate over time, it is possible that not all results arrive within 210 seconds at the client (see Sec. 4.5.5) if the call was issued in *explicit* mode, especially in tests with only 5% servers. Additionally, as the file size increases, the transmission time increases too, and the number of successful calls decreases as expected, as indicated by Fig. 4.30a, Fig. 4.30c, and Fig 4.30e.

4.5 DTN-RPC - Offloading Work in Challenged Environments

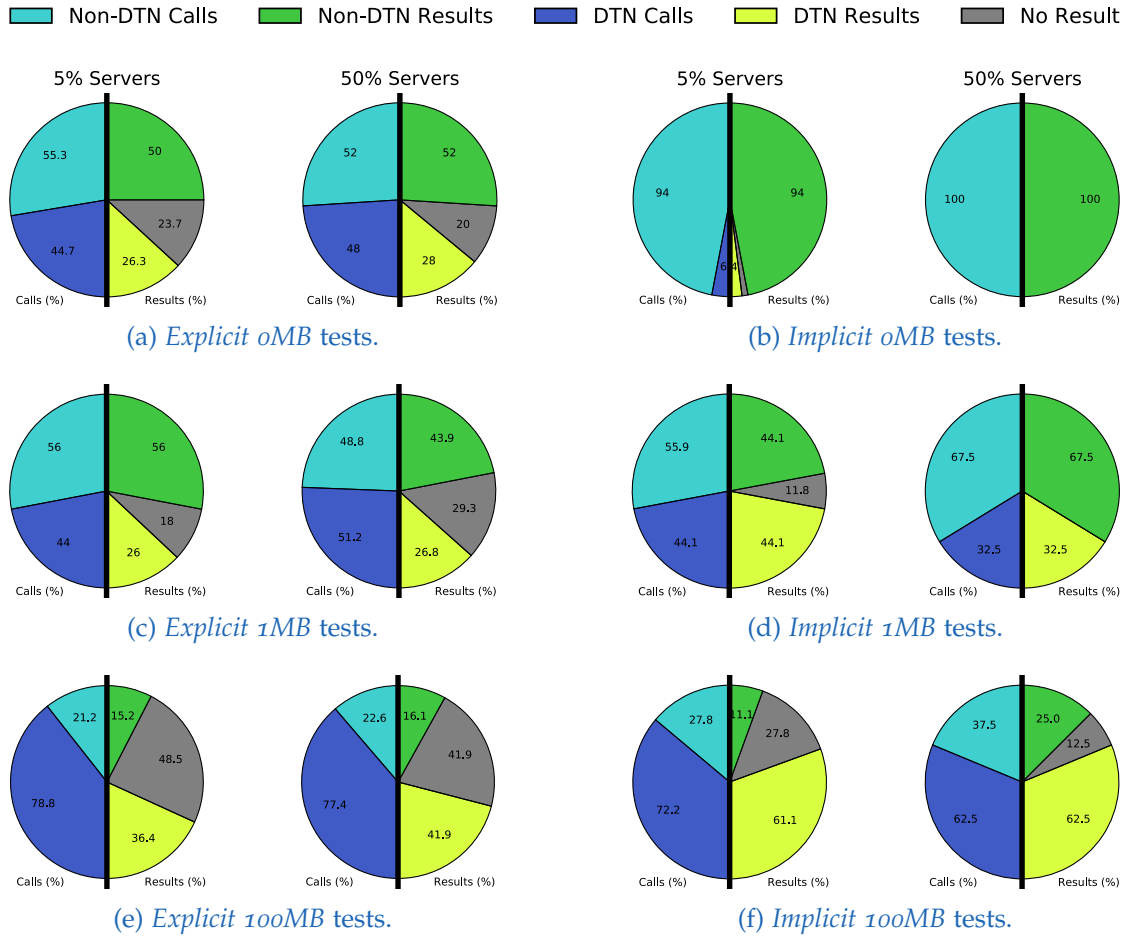


Figure 4.30: Percentages of procedures called and results returned via Non-DTN and DTN for 100MB in the Islands topology.

Furthermore, it is evident that some of the results arrive via MDP or MSP (i.e., Non-DTN), others only via Rhizome (i.e., DTN). There are two reasons. First, it is possible that a call is issued successfully using MSP, but the route from the server to the client gets lost because the islands have separated. Then, the result is sent via Rhizome and arrives after the islands have merged again. Second, the client cannot establish a connection to the server at all, because the islands are not connected. The procedure will be called using Rhizome and the client will wait via Rhizome for the result. Even if some results do not arrive in the *explicit* mode, the DTN protocol helps to improve the number of successful calls, as shown in Fig. 4.30. 41.9% of the results in the *explicit* tests with the 100MB test set with 50% of the nodes as servers arrive via Rhizome, and in 41.9% of the tests, no result arrives. In the *implicit* tests with the 100MB test set with only 5% of the nodes as servers, 61.1% of the results arrive via Rhizome, and only 27.8% of the results do not arrive at all.

Figures 4.30b, 4.30d and 4.30f show *implicit* tests in the Islands topology for three different file sizes with different numbers of servers. It is evident that the *implicit* mode increases the number of successful calls in every situation compared to the *explicit* tests. Due to the dynamically changing Islands topology and the relatively short contact

4 Disruption-tolerant Device-to-Device Emergency Communication

durations, it is still possible that not all results arrive in *100MB*. For the *explicit* calls, the more servers are available, the more results arrive.

The number of missing results can be decreased if the contact duration is increased or the waiting time for results is increased. Furthermore, more elaborate remote procedures require a lot more time to finish than the simple echo service used in this evaluation. Therefore, the waiting time for results of up to 210 seconds in the experiments should be increased in production environments, since it might be possible that a result arrives after hours at the client via DTN.

To summarize, the *transparent* mode helps to improve the probability of receiving results in dynamically changing network topologies like *Islands*. Furthermore, the *transparent* mode can deliver results where a traditional RPC would not lead to any response due missing network connections. Finally, if the waiting time for results is adequately large, the probability of receiving results increases, because when a DTN protocol is used, results do not get lost, but simply are not transmitted via a direct connection to the receiving node. Therefore, given sufficient time, results will always reach their destinations.

4.5.6 Conclusion

In this section, *DTN-RPC* was presented, a new approach to provide RPCs for DTN environments. *DTN-RPC* relies on (a) control and data channels to cope with potentially short contact durations in DTN where it is impossible to transmit large amounts of data, (b) explicit and implicit modes to address remote servers, (c) Non-DTN and DTN transport protocols for issuing calls and receiving results, and (d) predicates that servers check to decide whether a procedure should be executed. The implementation of *DTN-RPC* is based on Serval, an open-source, disruption-tolerant wireless ad-hoc networking system. The experimental results have indicated that the measured CPU and network overheads for *DTN-RPC* are reasonably low so that *DTN-RPC* can be executed on smartphones or routers, and that the round-trip times and the number of successful RPCs are highly satisfactory in dynamically changing network topologies with unstable links. Thus, *DTN-RPC* adds remote computing capabilities in the form of RPCs to DTN. These can, for example, greatly improve the tools available for professional responders during emergencies by utilizing low-power mobile devices that can offload tasks, such as requests for aerial overview images or for face recognition based comparisons to search for missing people. Furthermore, CPU-intensive tasks such as reconstruction of 3D models for replication of spare parts in the field [92] can be delegated off-the-grid to more powerful participants in the area.

There are several areas for future work. First, *DTN-RPC* has been tested and evaluated using emulated networks. We plan to perform tests with smartphones to get a better view on the real-world performance of *DTN-RPC* and a realistic evaluation of its energy consumption. Second, since the Non-DTN transport protocols produced satisfactorily results in the *Chained* and *Hub* topologies, *DTN-RPC* should be evaluated without relying on the strict differentiation between control and data channels. Finally, although it is relatively difficult to implement error handling and acknowledgment mechanisms, the evaluation has shown that this is not impossible. Thus, an acknowledgment system should be implemented for the *any* mode to inform other servers that the execution has already started.

4.6 Environmental Monitoring Platforms for Disaster Scenarios

4.6.1 Introduction

Environmental monitoring without being able to rely on existing electricity and communication infrastructures is required in several use cases, such as biological and ecological studies (e.g., animal tracking, air quality measurement), delivering Internet of Things (IoT) technology to farms⁵¹ in rural places, and emergency communication in disaster scenarios (e.g., tsunamis, earthquakes, nuclear meltdowns). In these situations, several constraints need to be considered when a technical solution for environmental monitoring is designed: lack of power supply, lack of communication infrastructures, harsh weather, low maintenance possibilities, weight restrictions for animal-attached sensors, availability of scenario-specific sensors, and cost factors.

There are several computing platforms readily available that can be extended to provide stationary services as well as mobile, low power tracking devices. Furthermore, there are many affordable, off-the-shelf sensors that are potentially useful in such scenarios, but proper information on how to integrate them and their specific requirements, such as real world energy consumption, are often not easily accessible. Moreover, integrating the computing power, flexibility and mobility of smartphones and tablets is an interesting option. Since satellite communication is expensive and limited, and cellular services are often not available in remote places or during a disaster, low-cost long-range (up to 16 km) radio technologies, such as LoRa, are quite useful. Although they are mostly associated with IoT applications in an infrastructure-based LoRaWAN mode, they can also be used for device-to-device communication and have the benefit of license-free frequency bands in any country of the world. Due to the low bandwidth of these radio transceivers, only small pieces of data can be transmitted, increasing the need to preprocess data prior to long-range transmissions.

In this section, a flexible and affordable sensor, computation, and communication platform for environmental monitoring that relies on low-cost hardware and infrastructureless communication is presented. It uses delay-/disruption-tolerant networking (DTN) for non-time critical tasks over different wireless links, provides on-device data processing capabilities based on machine learning methods, and integrates mobile sensor nodes, static devices, and smartphones. In particular, the following contributions are made:

- A novel sensor, computation, and communication platform for static and mobile environmental monitoring setups, and for users with mobile devices is presented.
- A novel energy-efficient approach for on-device image classification is presented.
- A novel approach to provide long range communication capabilities for Bluetooth-enabled devices is presented.
- Experimental evaluations of (a) commonly available and affordable platforms, (b) the power consumption of several sensors, and (c) the real world communication ranges and power demands of various radio link technologies for environmental monitoring are presented.

Parts of this section have been published in [6].

⁵¹<https://wazihub.com>

4.6.2 Related Work

Several wireless sensor platforms for environmental monitoring were presented in the literature, but they are often either based on specific technologies or tailored to dedicated use cases. For example, the OpenSense project aims to bring community-driven environmental monitoring to life with an open platform and accessible results [93]. However, it is specifically designed for air pollution monitoring, relies on existing communication infrastructures and hardware specifically built for this use case. On the other hand, it integrates static sensor nodes as well as mobile nodes, and most recently also wearables [94]. Similarly, Citi-Sense-MOB [95] and AirSenseEUR⁵² [96] work for air quality measurements in urban environments, but also rely on technologies such as GRPS for data transmission, constant power supply (e.g., from the bus or car the sensor is mounted on), and custom/closed-source printed circuit boards (PCBs).

Some custom-built platforms can be adopted to various scenarios. Problems associated with custom-built platforms are their cost and availability; sometimes they are specifically tailored to a particular geographic region [97], [98].

Llamas et al. [99] use Arduino and Raspberry Pi computers for building a reusable open sensor platform. Their application is human gait identification. Long range communication or infrastructureless operation are not considered.

The senseBox project⁵³ provides a sensor platform for citizen science projects [100]. There is a strong focus on education and publicly available sensor data, but senseBox lacks flexibility when used for more than just raw sensor data aggregation and also relies on existing communication infrastructures. Luftdaten⁵⁴ [101] also uses cheap microcontroller units (MCUs) configured for air quality measurement in a citizen science project, but also relies on existing communication infrastructures.

The EU project WAZIUP tries to bring IoT technologies to developing countries [102]. WAZIUP provides a low-cost communication infrastructure (LoRaWAN), while we try to work without any infrastructure on a purely peer-to-peer basis. We incorporate DTN technologies and feature heterogeneous radio-link setups. Furthermore, conserving energy plays a more vital role in our use cases, and we focus on providing a complete system for remote sensing and communication.

Some proposals use mobile devices or wearables as sensors, and some approaches provide low-cost, long-range radios usable from smartphones, but they are often very impractical (a direct USB connection to a smartphone is required) or require operating system modifications [103].

4.6.3 Sensor Platforms for Disaster Scenarios

Static Sensor Platforms (SSPs) are the backbone of our environmental monitoring platform. They have less restrictions on size or weight than Mobile Sensor Platforms (MSPs), and energy problems can be handled easier by adding solar panels, wind turbines, or larger batteries. SSPs can be used for relaying packets, collecting sensor data on roof-tops, on trees (e.g., wildlife cameras), or on smart street lamps. MSPs, on the other hand, are used to tag animals and, therefore, must be kept as light and small as possible. Energy is a more valuable resource. Since humans with smartphones and

⁵²<https://airsenseur.org/website/>

⁵³<https://www.sensebox.de>

⁵⁴<https://www.luftdaten.info>

4.6 Environmental Monitoring Platforms for Disaster Scenarios

tablets can also act as sensors, these must be integrated as well. Adding long-range infrastructureless communication to mobile devices also has the benefit that it can be used in case of an emergency for sending messages to other participants. Finally, it is necessary to preprocess the gathered sensor data to reduce the needed bandwidth for transmission. This is challenging when using power-efficient devices as a basis for SSPs. The different sensor platforms are discussed in more detail below.

Static Sensor Platforms

SSPs can easily utilize different energy sources, such as solar panels, car batteries, or electrical wall outlets. Therefore, they can be built using regular single board computers (SBCs), such as Raspberry Pi 3 or Zero, since these platforms offer a compromise between computational power and low energy consumption. To also function as a network hub for mobile users and MSPs, different radio link technologies can be incorporated - Bluetooth, WiFi mesh, and for longer range, license-free, low-bandwidth communication via LoRa. Since energy consumption of SSPs is not as critical as in MSPs, SSPs can constantly listen on the different wireless interfaces and act as hubs or relays for smaller mobile nodes passing by. For data dissemination, DTN technologies such as Serval⁵⁵ can be used, since they perform well in infrastructureless environments [2]. A variety of sensors can be added, e.g.:

- camera, night-vision camera, thermal camera
- microphone
- GPS/GLONASS
- temperature, humidity, barometric pressure
- air quality
- rain- and soil-moisture sensors

Usually, these are connected directly through GPIO pins or various bus systems (e.g., serial, SPI, i2c). Most SBCs directly provide these interfaces, only analog-digital pins are often lacking, but can easily be added externally.

Mobile Sensor Platforms

MSPs have more constraints than SSPs, since they must be as light and small as possible to be attached to animals or additionally mounted on an Unmanned Ground or Aerial Vehicles (UGV/UAV) [7]. Apart from the weight of the total system, power consumption is the main bottleneck. Due to these limitations, the choice of radio link technologies and the types of sensors are quite restricted. MSPs are based on small MCUs, since they require much less power and often provide deep sleep capabilities as well as various I/O pins for digital and analog sensor inputs.

Long-range Radio Links for Mobile Devices

In remote areas, cellular coverage or WiFi access points for communication using standard smartphones are often not available. Building custom radio links into phones is quite expensive and economically not interesting for large carriers. Therefore, a different approach with the following requirements is needed:

⁵⁵<https://github.com/servalproject/serval-dna>

4 Disruption-tolerant Device-to-Device Emergency Communication

1. it should work with any operating system,
2. it should be compatible with any mobile phone,
3. it should be license-free "worldwide",
4. it should provide infrastructureless long-range (>1 km) communication,
5. it should be energy-efficient,
6. it should be affordable.

The best solution for items 1 and 2 is to rely on platform-agnostic standards such as Bluetooth Low Energy (BLE) that can be used from Apple iOS and Google Android, or almost any other operating system. To cover items 3 and 4, we use the LoRa standard that in theory provides up to 16 km of range and is available in different frequency bands (433/868/915 MHz) for worldwide usage. LoRa also offers the LoRaWAN standard as a higher layer with built-in encryption, but it also requires some infrastructure, making it unsuitable for our task. Energy efficiency (item 5) is partly achieved by using BLE, but also by using small MCUs to handle communication. By finding a suitable MCU with built-in Bluetooth and LoRa chipsets, the price for such a smartphone add-on is also kept low. Alternatively, an SBCs equipped with a LoRa modem can be paired using Bluetooth with a smartphone. Even though power consumption will be higher, this system has the benefit that software such as Serval can run directly on the SBC and expose only UI relevant functions via BLE, preserving smartphone energy due to lesser notifications and wake-ups.

On-Device Data Processing

Since long-range links over LoRa only provide a few kilobits per second of bandwidth, transmitting large amounts of data in its raw form is not feasible. Therefore, processing the sensor data at least partially on the SSP to filter out unwanted content or already produce analysis results is favorable. In our use cases, we heavily rely on visual object/concept detection in images to filter out relevant information. Our previous work has shown how this approach can significantly reduce the amount of data necessary to be transmitted [5]. To cope with the limited CPU power of most SBCs, we integrate external hardware to accelerate the visual classification process. The challenge is to limit the power consumed by the accelerator device when it is currently not in use.

4.6.4 Implementation

Here, the implementation details of the developed platforms are discussed.

Static Sensor Platform

The SSP can act as a relay for MSPs and mobile devices carried by users. Furthermore, it can be used without any sensors as a base station and uplink to the Internet (see Fig. 4.31). To provide these features, we focused on ARM-based SBCs, more specifically the Raspberry Pi family. These devices are readily available worldwide, are proven technology made specifically for tinkerers and well documented. There are several cameras that can be directly attached, plus many (p)HATs with additional hardware and many GPIO pins for direct sensor attachment. Depending on particular requirements (size, energy etc.), there are different models available, such as the Pi Zero W and the Pi 3. Many wildlife cameras and weather stations have already been built on this proven



Figure 4.31: Base station for monitoring, relaying and processing.

hardware platform, and Raspbian provides a solid and familiar Linux foundation. We modified a Raspbian OS Image to provide features specific to our SSP. It can easily setup a mesh network, provide Bluetooth debugging capabilities, open an access point, and start services, such as GPS logging and serval-dna. Also, the energy consumption can be optimized by deactivating unused features, such as the HDMI port of the Raspberry. All this can be configured through a simple text file on the small FAT32 partition on the SD card⁵⁶. This has the benefit that the default image can be written to a new SD card and then can be configured from any computer with a simple text editor prior to actually booting the system. There is no need for Linux knowledge or extra drivers to read the filesystem, all is kept in one file, in one place for ease of use. The single biggest feature missing in the Raspberry Pi family is proper power management and deep sleep, but there are several after-market solutions^{57,58} offering this functionality. Another benefit of the newer Pi's (Zero W(H) and 3) is that Wi-Fi and Bluetooth are already present. The only major interface left to be added for our system is a LoRa transceiver for long-range communication.

LoRa Radio Modem There are several MCUs on the market using Cortex Mo, ATmega32u4, or ESP32 chips with onboard rfm95 transceiver modules. Using the RadioHead library⁵⁹, these can easily be used in the Arduino programming environment to send packet data from device to device. We developed a firmware to expose this functionality via an AT command set over the built-in USB-Serial, similar to the one found in classic dial-up modems. Therefore, any device with an USB port can use such a modem, and no special drivers are needed. The source of the modem firmware can be

⁵⁶<https://github.com/buschfunkproject/heckenschere>

⁵⁷<https://spellfoundry.com/product/sleepy-pi-2/>

⁵⁸<http://www.uugear.com/witty-pi-realtime-clock-power-management-for-raspberry-pi/>

⁵⁹<http://www.airspayce.com/mikem/arduino/RadioHead/>

4 Disruption-tolerant Device-to-Device Emergency Communication

found on github⁶⁰. Furthermore, Serval, our DTN middleware, was made aware of this communication channel by changing LBARD and writing a driver for our firmware⁶¹.

Mobile Sensor Platform

Since the MSP should be as light and small as possible, we focused on MCUs with integrated Wi-Fi and/or LoRa transceivers. For convenience, performance and portability, all of our programming was done using the Arduino programming environment in contrast to ESP's IDF or MicroPython. Hence, only minimal changes were necessary to switch from one MCU to another. To preserve battery power, the mobile sensors are mostly kept in deep sleep until a timeout or external trigger wakes them up. Also, they are programmed as send-only devices, since relaying data would require constant listening and would prevent deep sleep, thus draining the battery faster.

Long-range Radio Links for Mobile Devices

Since newer SBCs, such as the Raspberry Pi 3 and Pi Zero W, provide Bluetooth capabilities and can control devices with our radio modem firmware, such a setup can easily be used as a bridge for smartphones. For a simple prototype, we used the bleno framework⁶² to expose system functionality via BLE. This is also useful for debugging head-less systems. Therefore, we provide a simple echo service to broadcast any information via BLE⁶³ from our customized Raspbian distribution. Despite the flexibility this setup provides, it consumes quite a lot of power and is rather large in size (Pi Zero + LoRa modem + battery pack + antenna).

For a more lightweight solution, the radio modem firmware was enhanced to directly use BLE as a serial UART replacement. The packet size restrictions of LoRa and the BLE implementation in most RF95-based chipsets are quite similar. There are ESP32-based MCUs, such as the ones from Heltec and TTGO, that provide WiFi, Bluetooth and LoRa on a single board. The resulting system can be paired with any BLE capable (mobile) device, requires much less power, and for the 868/915 MHz bands, a rather short antenna. A modified version of the firmware can also be found online⁶⁴. This system is rather small and comparable to commercial products, such as the GoTenna⁶⁵, but more open and flexible. The BLE LoRa modem based on a TTGO ESP32 chip, including a small 750 mAh LiPo, a custom 3D printed case, and a 868 MHz antenna is shown in Fig. 4.32 right next to a Raspberry Pi 3 for a size reference.

On-Device Data Processing

For data processing on SBCs, a solution with two execution options was implemented. The first one uses the device CPU itself, and the second one utilizes a Intel® Movidius™ Neural Compute Stick (NCS). If we use the compute stick, the CPU of the SBC can go to deep sleep and save energy. Visual object/concept detection is used for

⁶⁰<https://github.com/gh0st42/rf95modem>

⁶¹<https://github.com/gh0st42/lbard-ng>

⁶²<https://github.com/noble/bleno>

⁶³<https://github.com/buschfunkproject/bledebug>

⁶⁴https://github.com/gh0st42/rf95modem/tree/ble_modem

⁶⁵<https://www.gotenna.com>



Figure 4.32: BLE LoRa modem with plain Raspberry Pi 3 for size comparison.

image processing, and for this purpose we use two neural network models. The first one is InceptionNet v3 [104] with 1001 different detectable classes, and the second one is also an InceptionNet v3 model trained on the Open Images Dataset⁶⁶ This neural network can detect 6012 different classes and is suitable for a more detailed analysis of the given images.

These neural network models are executed either with Tensorflow 1.1.0 built for ARM CPUs or on the Movidius compute stick. Additionally, the neural networks have to be converted with the Movidius compiler *mvNCCompile* to run the pretrained versions on the compute stick.

During our evaluation we found that the Intel Movidius NCS consumes an average power of 1955 mW when it is idle. On a Raspberry Pi, we can disable the entire USB subsystem, but in this case the Ethernet and all other USB ports are also disabled. Since various sensors and potential extra radio link interfaces might be connected over USB, this behavior is not desirable. Disabling only the specific port of the NCS to save energy when no processing has to be done is more favorable. To achieve this, *hub-ctrl*⁶⁷ is used to turn off the power of the compute stick in case it is currently not needed. This preserves a lot of energy, since the compute stick is expected to have more idle times than compute times. The possibility to have the compute stick as a backup without an energy penalty and only activating it when really needed makes it even more useful.

The visual object/concept detection software itself was written in Python 2.7 and outputs the five main visual concepts for each given image for further processing or discarding of irrelevant images. Thus, the energy for transmitting irrelevant images can be saved. Especially in a multi-hop scenario, a large amount of energy can be saved by applying preprocessing functions to recorded image data [19].

⁶⁶<https://storage.googleapis.com/openimages/web/index.html>

⁶⁷<https://github.com/codazoda/hub-ctrl.c>

4.6.5 Experimental Evaluation

Several experimental evaluations were performed regarding power consumption, transmission ranges, and execution times. The Monsoon Power Monitor was used for power measurements. All energy tests were repeated 2-3 times and ran for about one minute each, resulting in plenty of time to get a realistic power reading. Only the stress and compute stick evaluations were not terminated by time but by the given task.

Platform Comparisons

The evaluation includes some Raspberry Pi SBCs as well as MCUs from different vendors. They were tested for their idle power consumption and under full load. Other chipset-specific features such as RAM, onboard networking capabilities etc. are also listed in Table 4.7.

Table 4.7: Overview of SBC and MCU Platforms

	Processor	RAM	Network	Idle - Power in mW		Load - Power in mW	
				Mean	Max	Mean	Max
Pi 1 Model B+	1x 700 MHz	512 MB	100 Mbit Eth.	931	1288	1043	1418
Pi 3 Model B	4x 1200 MHz	1024 MB	100 Mbit Eth., WiFi, 4.1	1107	1845	2224	3285
Pi Zero W	1x 1000 MHz	512 MB	WiFi, 4.1	415	702	662	1057
ESP32, TTGO	1x 240 MHz	520 KB	WiFi, 4.2	366	427		
ESP8266, Feather Huzzah	1x 80 MHz	80 KB	WiFi	426	1321		
Feather Mo	1x 48 MHz	32 KB	Optionally WiFi, LoRa	25	29		
Feather 32u4	1x 8 MHz	2 KB	Optionally LoRa	24	27		
Waspote	1x 14.74 MHz	8 KB		145	150		

When looking at the different Raspberry Pi models, it is evident that when idling, the Pi Zero W (415 mW) consumes less than half of what a Pi 1 (931 mW) needs and just about a third of what the base Pi 3 Model B (1107 mW) needs. The newest Pi 3 Model B+ was not tested, but due to its increased CPU power it is expected to consume even more power. Both Pi 1 and Pi 3 offer more USB ports onboard and Ethernet. The Pi 3 offers more processing power and cores than the relatively power-hungry Pi 1 that provides even less processing power than the Pi Zero W. Under load, the Pi Zero W still consumes less power than a Pi 1 idle. Due to its 4 cores and high clock speed, the Pi 3 uses 2224 mW under load and peaks to 3285 mW at certain times. Thus, if saving power is the priority, then the Pi Zero W is the best choice. If multiple CPU cores, onboard Ethernet, and more USB ports are essential, then the Pi 3 Model B is the best candidate.

For the MSP several MCU candidates were evaluated, as shown in the lower part of the table. Some of them like the ESP32 board from TTGO are much more powerful with 240 MHz and 520 KB RAM compared to systems like the Waspote with 14.74 MHz and 8 KB RAM. Not all systems provide deep sleep functionality by default. Even though the Waspote or Feather Mo system consume less power than the ESP32, they also lack the radio link interfaces. Due to the flexibility through the included radio interfaces, cost of the board and provided CPU/RAM as well as deep sleep capabilities and I/O pins, the MSPs were mainly built on ESP32 boards. Should RAM not be an issue and a single radio-link interface be sufficient, the Feather Mo or even the low-power 32u4 are good alternatives with minimal power requirements.

Evaluation of Sensor Requirements

The typical power requirements of different sensor types were measured, as shown in Table 4.8. The sensors range from simple temperature and particle sensors over GPS receivers to complex optical systems such as (night vision) cameras.

Table 4.8: Overview of different Sensors

	Function	Power in mW	
		Mean	Max
MAX30105	Particle	19	145
CCS811	VOC, TVOC, eCO ₂	52	186
PIR	PIR	51	55
SIM28 GPS	GPS	250	302
BME280	Temp., Baro., Humidity	56	65
MICS-2714	NO ₂	42	45
MICS-5524	CO	41	60
AMG8833	IR thermal	179	2248
Envirophat	Temp., Baro., Color, Accel., Magnetometer	180	1868
Pi Camera NV	Night Vision Camera	1173	2290
Pi Camera v2.1	Camera	393	2231

Most of these sensors consume an average power of about 50 mW or less. A notable exception is GPS with about 250 mW and the cameras with 393 mW / 1173 mW. It is also noteworthy that the night vision camera draws 3 times as much power as the regular one. A simple 8x8 thermal imaging sensor, such as the AMG8833, on the other hand, only consumes an average of 179 mW. When adding these sensors to an MSP or SSP, the maximum peaks must also be considered. These can go up to 2290 mW for the optical sensors.

Radio Link Comparisons

We evaluated the power consumed by different MCUs with onboard radio transceivers, ranging from Wi-Fi to LoRa, as shown in Table 4.9. This table shows the power consumption from the MCUs with included radio transceivers. It is obvious that Wi-Fi communication is at least twice as expensive as LoRa (TTGO ESP32), and in case of the Feather Mo, about 10 times as expensive. There is also a bigger difference regarding LoRa power consumption when comparing both Feather devices (ca 46 mW) to the TTGO ESP32 (235 mW). Furthermore, sending via LoRa is much more expensive at roughly 528 to 816 mW. If more RAM and CPU power is needed or if more than one radio transceiver with small physical dimensions is required, then the TTGO is the best choice. The Feather sticks are available only either with Wi-Fi or LoRa onboard.

To evaluate the LoRa transceiver modules' communication range, we deployed our sensor box (Fig. 4.33) and measured the transmission range using one of our GPS modules. We used ESP32 TTGO modules in various configurations. One chip was tuned to the 433 MHz band and equipped with a small wire antenna from the supplier. The next one was set to work in the 868 MHz band, also with the short vendor-supplied

4 Disruption-tolerant Device-to-Device Emergency Communication



Figure 4.33: Waterproof static sensor box deployed.

Table 4.9: Power consumption of MCUs including radio transceivers

	Power in mW	
	Mean	Max
ESP32, TTGO, WiFi	591	1353
ESP8266, Feather Huzzah, WiFi	405	2118
Feather Mo, WiFi	475	1577
ESP32, TTGO, LoRa	235	816
Feather Mo, LoRa	45	528
Feather 32u4, LoRa	47	568

antenna. Finally, we set up another TTGO in the 868 MHz band but on a different channel with larger 3.5 dBi magnetic sucker antennas on both ends. The sending interval chosen for the fixed station was set to 7 seconds for each device.

The city of Marburg is surrounded by many hills and forests, therefore, the sending station was deployed on a viewpoint near the university with line-of-sight to most parts of the city. The receiving nodes were mounted on the dashboard of a car, and we drove around the campus, through the city, up to the castle on the opposite side of the city, and then along the valley, until we lost the signal. The route we took, including received beacons, is shown in Fig. 4.34.

The longest distance covered was almost 6.5 km with the 3.5 dBi sucker antenna. The 433 MHz setup and the other 868 MHz node reached 1-2 km less, but with a much higher packet loss. Also, there was no more line-of-sight from either maximum distance points, it was blocked by hills, houses and trees. The 1.5-2 km distance to the downtown area was covered by all three devices despite trees and houses in the way.

The RSSI over distance is shown in Figure 4.35, where up to 2-3 km all devices still received signals quite often. As expected, the 3.5 dBi antenna has the best reception throughout the test. This is especially clear for distances over 3 km where the duck antennas both fall short. The total number of received packets in percent can be seen in Fig. 4.36. The clear winner here, again, is the 3.5 dBi sucker antenna. Both stock

4.6 Environmental Monitoring Platforms for Disaster Scenarios

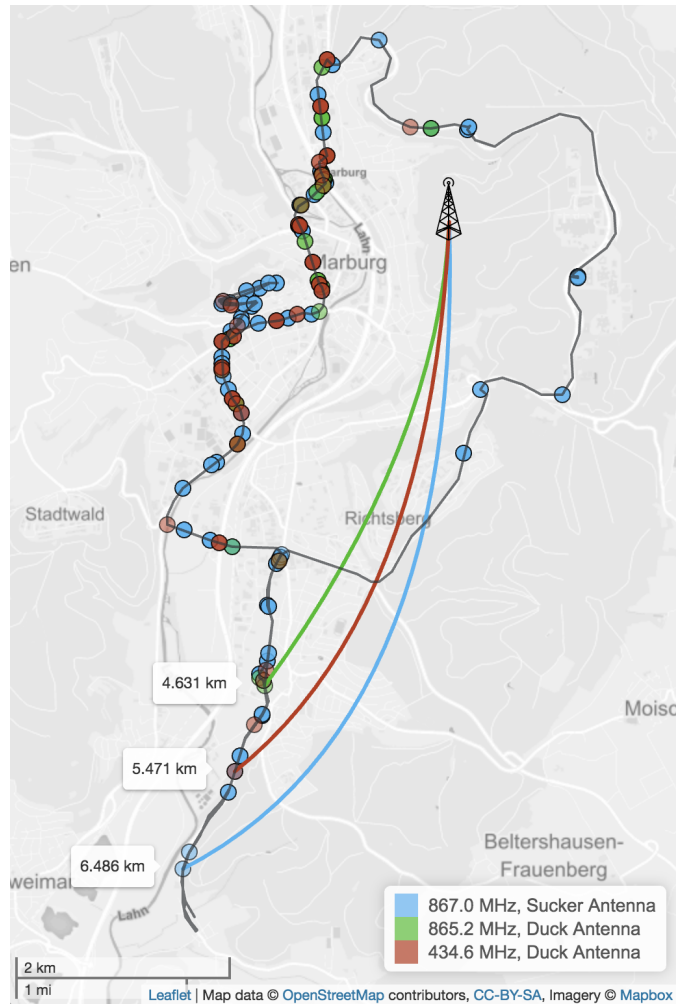


Figure 4.34: Communication range of different LoRa setups.

antennas from TTGO perform very similar with reception rates between 10% and 15% during the test. Due to our constant movement and the diverse topographic areas, these numbers should be seen in relation to the sucker antenna, not as absolute numbers. Traffic might be responsible for remaining longer in areas where no reception was possible, resulting in a higher overall packet loss rate.

On-Device Data Processing

For on-device data processing, we used 1481 images for CPU and neural compute stick execution. Each of these images is 3000 x 2000 pixels in size, since 6 mega-pixels are a good compromise between storage space and visual details. This dataset is described in our previous work [5]. For evaluation purposes, we built a Python tool to batch process these images. First, the program loads the necessary libraries and the pretrained neural network, then the images are processed. The power consumption and time for the tasks is shown in Table 4.10.

First, we compared the idle power consumption of the Pi 3 to one with an attached neural compute stick. While the max peak is almost identical, the average power

4 Disruption-tolerant Device-to-Device Emergency Communication

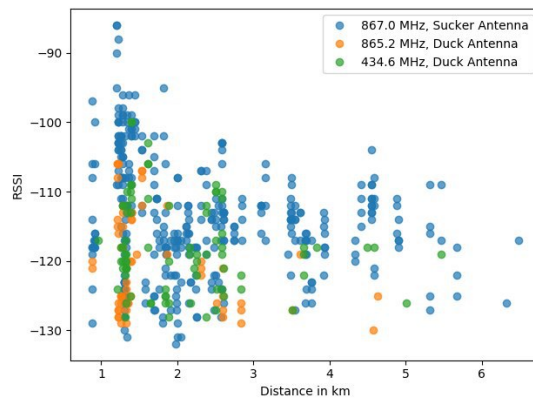


Figure 4.35: RSSI vs. Distance

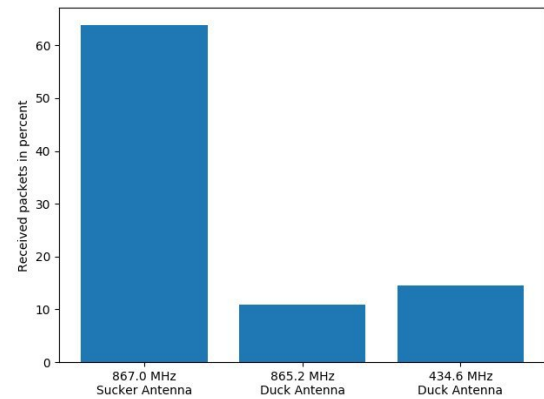


Figure 4.36: Number of received packets in relation to packets sent.

Table 4.10: Comparison of Concept Detection on Pi vs. Neural Compute Stick

	Idle - Power in mW			Starting - Power in mW			Analysis - Power in mW		
	Mean	Max	Time	Mean	Max	Time	Mean	Max	Time per Image
Pi 3b, 1K, CPU	1107	1845	59.29 sec.	2626	2895	19.66 sec.	2723	3537	13.27 sec.
Pi 3b, 1K, Movidius	1955	2095	59.76 sec.	2372	2884	3.92 sec.	2760	3388	0.61 sec.
Pi 3b, 6K, CPU	1107	1845	59.29 sec.	2659	3203	75.03 sec.	2871	3537	5.46 sec.
Pi 3b, 6K, Movidius	1955	2095	59.76 sec.	2507	2879	12.64 sec.	3313	3667	0.82 sec.

consumed is nearly double when a NCS is added. In the starting phase, the neural network is loaded either into RAM for the plain Pi 3 or onto the NCS. Both setups draw around 2500 mW with a slight edge for the NCS setup. Taking runtime into account, the compute stick is clearly more efficient by loading the trained model 5-6 times faster. When performing the actual visual concept detection, the mean power is between 2723 and 3313 mW and peaks at 3667 mW, with a slightly lower power demand by the CPU setups. Given the much longer runtimes of the CPU version, 6x and 21x slower per image, the NCS is still much more efficient, in terms of speed and energy. The main drawback of the tested compute stick is its high power consumption when idle.

This problem was solved by deactivating the specific USB port when the stick is currently not needed, reducing the power consumption overhead to zero.

Setup Cost

Since we want to provide affordable solutions that can easily be customized to specific needs, we also provide a short overview of the cost of our systems. All prices are in (rounded) Euro, taken in June 2018 from Amazon, Aliexpress etc. All our cases are custom printed on a 3D printer. We also did successful deployments with cheap waterproof junction boxes from the local hardware store (<10 €).

The cheapest system is our BLE LoRa Modem with a total system cost of about 14 € (Table 4.11). A 750 mAh battery lasts roughly between half a day and a full day. This component can easily be swapped for a bigger battery or powerbank.

4.6 Environmental Monitoring Platforms for Disaster Scenarios

Table 4.11: Cost of BLE LoRa Modem

Part	Estimate Price	Comment
TTGO LoRa SX1276 ESP32	9 €	incl. antenna
LiPo Battery 750mAh	3 €	
3D printed case	2 €	
Total Price	14 €	

The MSP example shown in Table 4.12 is usable for GPS tracking of larger animals plus some added sensors for collecting weather data. In real world deployments with a 750 mAh battery and gathering samples every 15 seconds, the power lasted for about 6 days. With a reasonably lightweight and cheap 1500 mAh battery and a lower sampling rate, we can achieve runtimes over two weeks. The total cost for this setup is around 46 € per device, mainly dominated by the price of the SD card for data logging and the GPS module.

Table 4.12: Example configuration for a MSP

Part	Estimate Price	Comment
TTGO LoRa SX1276 ESP32	9 €	incl. antenna
LiPo Battery 1500mAh	3 €	
3D printed case	4 €	
Ublox NEO GPS	12 €	incl. antenna
SD-Card Breakout	2 €	
32 GB SD Card	13 €	
Real Time Clock	1 €	
Temperature & Humidity Sensor	1 €	
Gyro Sensor	1 €	
Total Price	46 €	

In Table 4.13, an SSP example, used for relaying data as well as actively gathering sensor data and processing it on device, is shown. Depending on whether a compute stick is used or not, the cost varies between 135 € and 201 €. The cost for a blank relay-only system would be around 65 €. The rest is the cost of the added sensors and the NCS. What this setup is lacking, is a power supply. Depending on the location of deployment, a battery system (20 € - 200 €) and/or a solar panel (50 € - 200 €) could be added if a direct power supply is not available.

4 Disruption-tolerant Device-to-Device Emergency Communication

Table 4.13: Example configuration for a SSP (w/o power supply)

Part	Estimate Price	Comment
Raspberry Pi 3	32 €	
TTGO LoRa SX1276 ESP32	9 €	incl. antenna
3D printed case	10 €	
Ublox NEO GPS	12 €	incl. antenna
32 GB SD Card	13 €	
Real Time Clock	1 €	
Temperature & Humidity Sensor	1 €	
Pi Camera	20 €	night vision opt.
<i>Movidius NCS</i>	66 €	<i>optionally</i>
Thermal Imaging	32 €	
Rain Sensor	1 €	
Soil Moisture	3 €	
PIR Motion Detector	1 €	
Total Price	135 € / 201 €	

4.6.6 Conclusion

In this section, a flexible and affordable sensor, computation, and communication platform for environmental monitoring was presented. Solutions for static and mobile setups, and integrated mobile devices, such as smartphones, into long-range infrastructureless radio networks were provided. Apart from evaluating typical power requirements of common hardware platforms and sensors, LoRa radio transceivers were also included in the study and evaluated regarding their realistic communication ranges. Furthermore, to make optimal use of these long-range, low-bandwidth links, a solution to on-device preprocessing of image data using neural compute sticks was provided in an energy efficient and computationally fast way. Finally, the expected costs of the subsystems used in the evaluation were presented. The components developed for the setups, such as the rf95 modem firmware, the BLE modem, Serval LBARD support, and the software customized for the Raspbian distribution, are all released as open source software on github.

In the future, the possibilities of direct device-to-device communication via cheap LoRa addons for smartphones should be used more extensively. This opens possibilities for new apps on mobile devices, such as infrastructureless messaging, or using a phone for sensing and remote control tasks. Furthermore, the usefulness of DTN software directly on the MCUs based on miniDTN should be explored. Finally, the integration of energy harvesting solutions would also increase the flexibility of the system when deployed in remote places.

4.7 Applications for Disaster Response: SmartFace

4.7.1 Introduction

During many natural or man-made disasters, a common task is to find missing persons. For this purpose, Facebook and Google offer services such as *Safety Check*⁶⁸ or *Person Finder*⁶⁹, respectively. However, both services require a working Internet connection and thus cannot be used when telecommunication infrastructures of mobile phone operators fail. In such emergency situations, smartphones, tablets and/or battery powered wireless routers can be used alternatively to spontaneously establish a disaster-response communications network to share data in a peer-to-peer fashion.

To support the search for missing persons, photos taken by mobile device users staying inside a disaster area can be spread around using disruption-tolerant networking (DTN) [56]. Since CPU power, memory space, and battery capacity of mobile devices are limited, the size of the photographic image data can be reduced by detecting persons' faces in images and transmitting only the extracted faces. In many cases, faces only make up a small fraction of a complete photo, and only sharing these over a wireless on-demand emergency network would significantly save resources.

For desktop and server systems, several face detection libraries with different properties in terms of recall and precision are available, but their resource requirements do not match with the resources of today's mobile devices, since the latter still only provide a fraction of the computing power offered by a contemporary workstation.

In this section, a novel approach, called *SmartFace*, to perform face detection *in situ* on smartphones or tablets in an efficient manner is presented. The approach is based on a novel two-stage combination of state-of-the-art face detection algorithms, further enhanced by region of interest selection, color space/depth reduction, resolution scaling, face size definition, image scaling, image cropping, and bounding box scaling. *SmartFace* improves the face detection rates within the same runtimes or obtains the same face detection rates within faster runtimes compared to the individual face detection algorithms used alone, and also reduces the amount of data that needs to be stored on disk and sent over the network. As a consequence, the battery life of mobile devices is extended, too. The main contributions are:

- A novel two-stage processing pipeline for resource-efficient face detection on mobile devices, with improved overall face detection rates and runtimes.
- An experimental study of image preprocessing parameters to obtain algorithm agnostic speed gains.
- Methods to reduce our original image data set by a factor of 133, indicating significant savings of network bandwidth and disk storage resources.

Parts of this section have been published in [5].

4.7.2 Related Work

Emergency communication networks [49] typically rely on radio technologies, such as Bluetooth, LoRaWAN, WiFi, TETRA digital radio or satellite links. They either

⁶⁸<https://www.facebook.com/about/safetycheck/>

⁶⁹<https://google.org/personfinder/global/home.html>

4 Disruption-tolerant Device-to-Device Emergency Communication

use telecommunication infrastructures, are distributed (peer-to-peer, mobile ad-hoc networks), or form hybrid architectures of both.

Several approaches utilize commodity mobile devices to realize hop-to-hop DTN [28], [56], [57]. Due to their peer-to-peer nature, they are well suited for (post-)disaster scenarios. Projects such as FireChat⁷⁰, Briar⁷¹, Serval⁷² and Forban⁷³ transfer messages and files between heterogeneous devices and share them locally through store-and-forward technology.

Furthermore, several face detection algorithms have been proposed. Viola and Jones [105] offer an approach in *OpenCV* that accelerates face detection. The tradeoff is that with an increased detection rate, the false positive rate also increases.

Felzenswalb et al. [106] present an object detection system for *dlib*. Since a sliding window over an entire image in different resolutions is used, the runtime increases with increasing image size. Therefore, small interesting regions within an image should be selected before the algorithm is applied.

Cheney et al. [107] perform a comparison of face detection algorithms. The authors state that often used test sets, such as Labeled Faces in the Wild (LFW)[108] and YouTube Faces [109], do not pose any challenges for current algorithms and should not be used any more for benchmarking. Instead, they use the IARPA Janus Benchmark-A face challenge (IJB-A)[110] that consists of over 5000 images and 20,000 video frames for benchmarking. The main open source algorithms highlighted in their paper are from *Dlib* and *OpenCV*, the first one with good detection rates and the second one being one of the fastest.

The typical approach to perform face detection on mobile devices is to offload images to a server and run a face detection algorithm on the server [111], [112]. Using powerful servers makes it much easier to achieve good face detection rates, but in an emergency scenario such servers are often not available.

Feng et al. [113] present a cascaded classifier approach that has also been tested on a Samsung S6 smartphone, resulting in a runtime of 34 ms for detecting faces in a 640 x 480 pixel photo. Compared to current 8 or 16 megapixel cameras in today's smartphones, this size is very small, and no public code is available to verify the results.

4.7.3 SmartFace's Design

The goals of *SmartFace* are as follows:

- Since the recognized faces have to be stored on mobile devices and transferred over wireless links, the size of the required data should be minimized. This also means that the false positive rate of the used face detection approach should be as low as possible.
- Since a data transfer can only happen after face detection has finished, the runtime of the used face detection approach should be minimized. This also means that the false negative rate and the false positive rate of the used face detection approach cannot be minimized at the same time, but the false negative rate should be as low

⁷⁰<http://opengarden.com/firechat>

⁷¹<https://briarproject.org/>

⁷²<http://www.servalproject.org>

⁷³<http://www.foo.be/forban/>

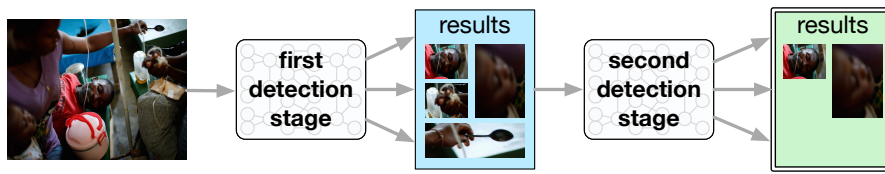


Figure 4.37: Basic two-stage face detector

as possible in accordance with the available computational resources on mobile devices.

To achieve these goals, the basic idea is to develop a two-stage processing pipeline in which the first stage is responsible for quickly selecting interesting regions in a given image, and the second stage performs in-depth face detection and validation in the interesting region selected in the first stage (see Fig. 4.37). Thus, the first stage is optimized for speed, favoring high recall over high precision. The second stage is optimized for quality, favoring high precision over high recall. By using adequate face detection algorithms that support the competing objectives of each stage, overall speed and quality improvements can be achieved.

This two-stage face detection approach can be flexibly configured to the preferences of a particular usage scenario, e.g., (a) by reducing the runtimes to save battery power, (b) by improving the quality of the face detection results by running the algorithms for a longer time, or (c) by trying to get the best results in a fixed amount of time.

There are several parameters that can change the runtime and the quality of the results of a face detection algorithm without changing the algorithm itself. To support the goals of *SmartFace*, the following set of parameters and their values are considered:

Color space: color / greyscale / black-and-white

Color depth: 24-bit / 8-bit / 1-bit

Resolution: $100 \leq n \leq 5000$ in steps of 100 pixels

Image scaling: area, cubic, lanczos4, linear, nearest interpolation

Cropping: remove from each border 0%, 10%, 20%, 30% of the full image

Face size: min: 80 x 80 pixels; max: 1000 x 1000 pixels

Bounding box scaling: $50 \leq n \leq 500$ in steps of 50

Some of these parameters are relevant for operations in a preprocessing step, while other parameters are applied to the first or second detection stages, respectively, as shown in Fig. 4.38. For example, having a larger bounding box in the second stage might increase the number of results, but might overlap with nearby faces in the same region. Here, duplicates have to be detected and eliminated. This is achieved by allowing bounding boxes to overlap to a certain degree after the second detection stage.

4.7.4 Implementation

The implementation of *SmartFace* is based on three main components. The first component handles the parameters and the preprocessing steps. The second component is a fast face detection algorithm for the first stage of *SmartFace* to select regions of interest. The third component is a high-quality face detection algorithm for the second stage of *SmartFace* to operate on the selected regions of interest.

4 Disruption-tolerant Device-to-Device Emergency Communication

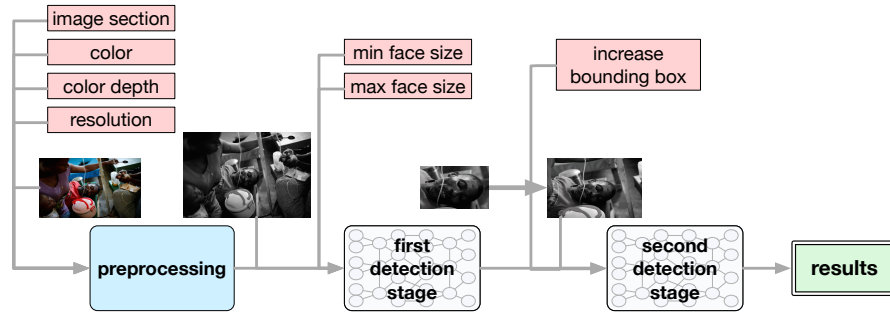


Figure 4.38: Two-stage face detector, preprocessing, and parameters

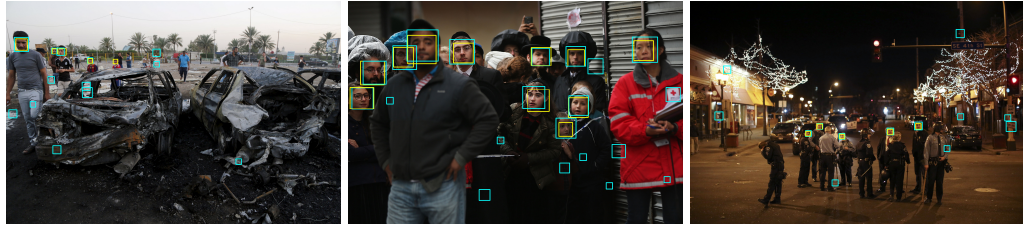


Figure 4.39: SmartFace in action

OpenCV's Viola/Jones algorithm [105] is used in the first stage, since it is a common out-of-the-box solution for face detection, in contrast to *OpenCV's* SURF that is also suitable for general object detection. It is comparatively fast [107], but has a relatively high false positive rate (which is acceptable as a pre-filter), as shown in Fig. 4.39 by the blue bounding boxes in each image. *Dlib* is used in the second stage. It is slower than the Viola/Jones algorithm, but has a lower false positive rate and a higher precision [107]. The faces detected by *dlib* are shown by the yellow bounding boxes in Fig. 4.39.

All parameters except the face sizes are independent of the used face detection algorithms. Therefore, the algorithms can be replaced by alternatives, still benefiting from the rest of the optimizations. The entire program flow is shown in Figure 4.40.

To determine suitable parameters for the scenario, a parameter scan by creating a test environment in *Bash* and using the created API to automatically iterate over the parameter sets using the image test set is performed. The parameter scan is first executed with *dlib* to match the results to the behavior of *OpenCV*. The results are presented in Section 4.7.5.

4.7.5 Experimental Evaluation

Test Environment

Three different devices are used in the experimental evaluation, as shown in Table 4.14: (a) a workstation as a reference platform, (b) a high-end smartphone (OnePlus 3T), and (c) an older mid-range tablet (Nexus 7). The workstation operates under Ubuntu 16.04 LTS, and both Android devices use the latest Android release 6.0.1.

4.7 Applications for Disaster Response: SmartFace

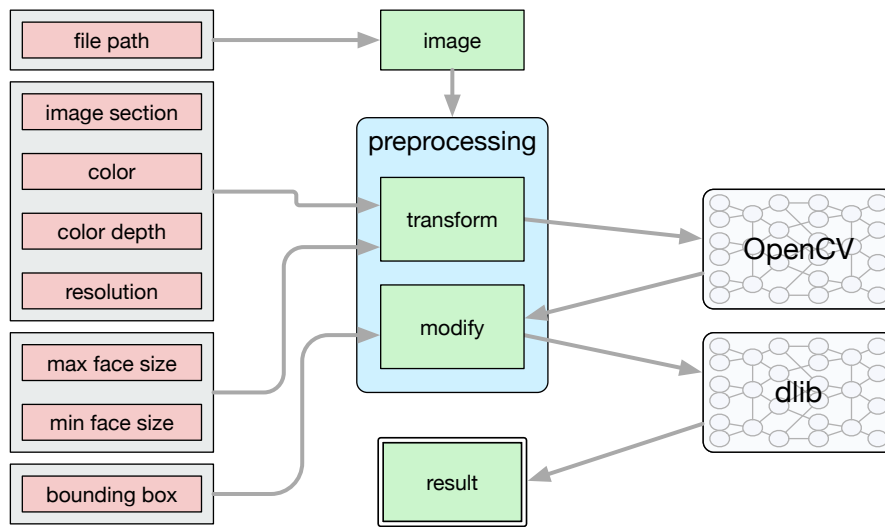


Figure 4.40: SmartFace implementation

Table 4.14: Device Specifications

Device	CPU	RAM	Storage
Workstation	Quad-core (i7-2600 @3,4GHz)	8GB	256 GB SSD
OnePlus 3T	Quad-core (2x2,35 GHz & 2x1,6 GHz)	6GB	128 GB Flash
Nexus 7	Quad-core (4x1,5GHz)	2GB	32 GB Flash

Image Test Set

To evaluate *SmartFace* on realistic images taken during emergency scenarios, an own test set by randomly was created by downloading images from the Internet (two examples are shown in Figure 4.41) using one of the scenario specific search terms from the following list: *haiti earthquake*, *haiti earthquake people*, *haiti earthquake faces*, *earthquake faces*, *earthquake people*, *disaster people*, *disaster faces*, *disaster management*, *flooding people*, *flooding faces*, *natural hazard people*, *natural hazard faces*, *tsunami people*, *tsunami faces*, *night bushfire*, *bushfire people*, *firefighter disaster*, *explosion people*, *accident people*, *syrian civil war*, *crowd*, *crowd disasters*, *crowd control*, *crevices earthquake*.

A crawler for *Google Images* was developed and it downloaded the top 100 search results, finally obtaining a total of 2,400 images. Duplicates were removed prior to usage. The crawler downloads images with a resolution of 3000 x 2000 pixels. Thus, these images correspond to photos taken by a 6 megapixel camera. Current smartphones can take photos with a higher resolution, but in a typical disaster scenario it is not realistic to assume that top-notch hardware is commonly available. Thus, 6 megapixel photos are quite realistic and much more sophisticated than commonly used face detection test sets, such as LFW where images have a resolution of 250 x 250 pixels, and a large image part is covered by a face.

After removing duplicates, the test set consists of 1,481 images. This leads to 2.8 GB image data that needs to be processed in the benchmarks. In addition, the faces for each image in our test set were manually labeled. It contains 2,419 faces to be detected. A list

4 Disruption-tolerant Device-to-Device Emergency Communication



Figure 4.41: Examples from image test set

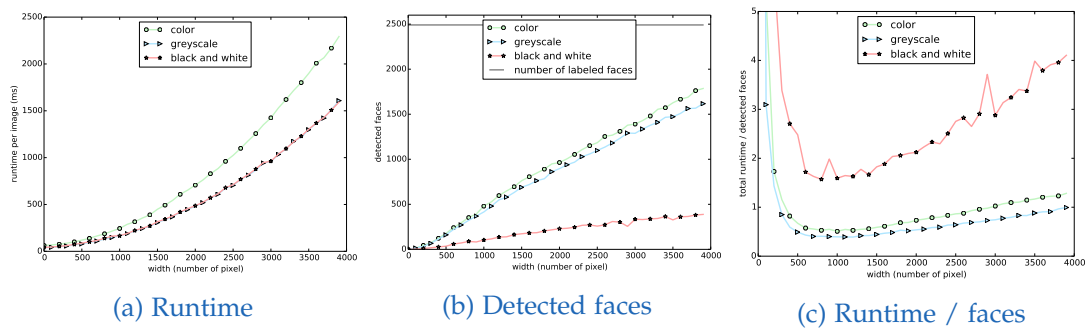


Figure 4.42: Comparison of different color spaces and depths.

with 2-D coordinates is stored for each image in the test set. These coordinates represent a central point in the face, in many cases this is the tip of the nose. Furthermore, the rule suggested by Cheney et al. [107] to only mark faces where both eyes can be seen was followed.

Parameter Scan

To evaluate the selected parameters, the results achieved with *dlib* are presented below; each preprocessing experiment is repeated 10 times. The results using *OpenCV* are similar.

Color Space and Depth In this experiment, the 24-bit input image is reduced to an 8-bit greyscale image and to a 1-bit black-and-white image. As indicated by Fig. 4.42a, the runtimes for greyscale and black-and-white images are nearly identical, and both are faster than for the fully colored image. This effect increases with increasing image size. Changing color space and depth also affects the detection rates, as shown in Fig. 4.42b. Using a greyscale image produces almost as good results as using the original image, but black-and-white yields significantly worse results. Fig. 4.42c shows the combination of the first two graphs; lower numbers are better. Again, greyscale and full color images are similar in their score, with a slight edge for greyscale due to its runtime being faster than its decrease in detection rate.

4.7 Applications for Disaster Response: SmartFace

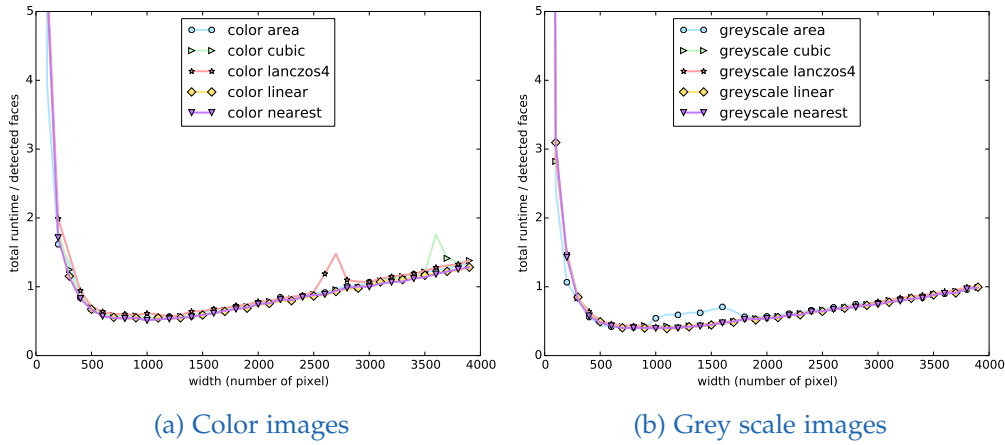


Figure 4.43: Number of detected faces per detection time for different resolutions.

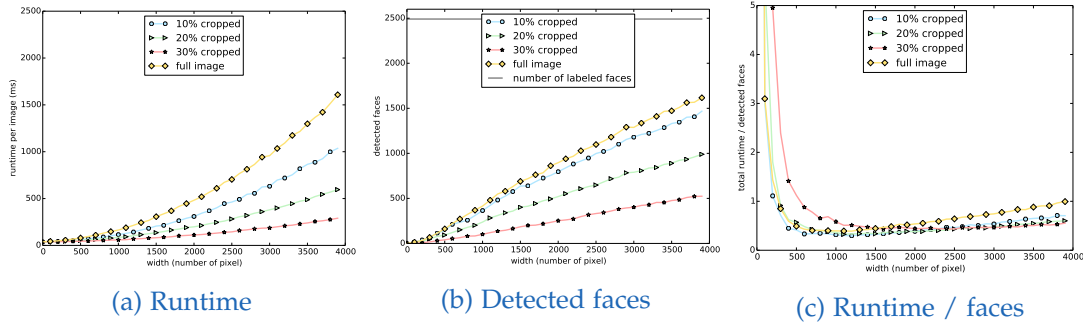


Figure 4.44: Comparison of cropped areas from grey scale images.

Image Scaling These experiments are performed on the original and greyscale images; black-and-white is not considered due to the results from the color space tests. Although the most common interpolation mechanisms for scaling images such as *linear*, *area*, *cubic*, *nearest* and *lanczos4* produce slightly different images, they are all comparable in terms of runtime and number of detected faces (see Fig. 4.43).

Image Cropping The next parameter evaluated is cropping the greyscale image. For each image, 0%, 10%, 20% and 30% are cut off from each border, reducing the image size, but also reducing areas where possible faces could be detected. As expected, image cropping reduces runtimes, even more when an original image is upscaled, as shown in Fig. 4.44a. Cropping the image by 30% from each border significantly reduces the number of detected faces, while cropping 10% decreases the detection rate only slightly (see Fig. 4.44b). Combining both scores shows an interesting result in Figure 4.44c where no single strategy wins. The behavior changes between 1,500 and 2,000 pixels. Cropping around 10% gives the most reliable results independent of image size. However, since not many portrait photos can be expected in our emergency scenario, one cannot assume that humans are always centered in the images, therefore cutting off the edges might miss important information.

4 Disruption-tolerant Device-to-Device Emergency Communication

Table 4.15: Contingency table for *dlib*

	faces detected	undetected	
labeled faces	1,194	1,226	Recall (R): 0.49
unlabeled areas	178	—	
	Precision (P): 0.87		F1-Score: 0.63

Table 4.16: Contingency table for *OpenCV*

	faces detected	undetected	
labeled faces	1,967	502	Recall (R): 0.79
unlabeled areas	6,106	—	
	Precision (P): 0.24		F1-Score: 0.37

Face Detection Comparison

Next, the face detection quality of the *dlib* and *OpenCV* algorithms are compared. Tables 4.15 and 4.16 show that *dlib* is superior with an overall F1 score of 0.63 compared to the 0.37 of *OpenCV*. However, the recall value of 0.49 for *dlib* is much lower than the recall value for *OpenCV* with 0.79, whereas the precision of *dlib* with 0.87 clearly outperforms *OpenCV* with a precision of 0.24. This confirms that *OpenCV* is well suited for the first stage of *SmartFace*, while *dlib* is an ideal candidate for the second stage.

Table 4.17 shows the results for combining both algorithms within *SmartFace* where it was tried to run *SmartFace* as fast as possible to obtain roughly the same number of detected faces as *dlib*. In contrast, in Table 4.18 *SmartFace* was given the same amount of time as *dlib*, leading to a higher precision of 0.93 and also a higher recall of 0.70 compared *dlib*. This also results in a higher F1 score of 0.80 compared to 0.63 of *dlib*.

Table 4.17: Contingency table for *SmartFace* (faster runtime)

	faces detected	undetected	
labeled faces	1,172	1,263	Recall (R): 0.49
unlabeled areas	238	—	
	Precision (P): 0.83		F1-Score: 0.61

Table 4.18: Contingency table for *SmartFace* (higher quality)

	face detected	undetected	
labeled face	1,741	732	Recall (R): 0.70
unlabeled area	129	—	
	Precision (P): 0.93		F1-Score: 0.80

Compared to using color or greyscale images in *dlib*, *SmartFace* is up to two times faster depending on the image size, as shown in Figure 4.45a. The overall number of detected faces is only slightly smaller, particularly when compared to standard greyscale

4.7 Applications for Disaster Response: SmartFace

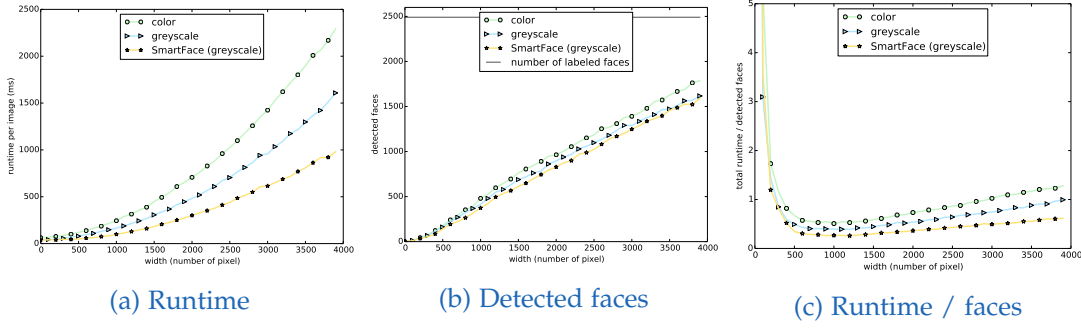


Figure 4.45: Overall benchmark

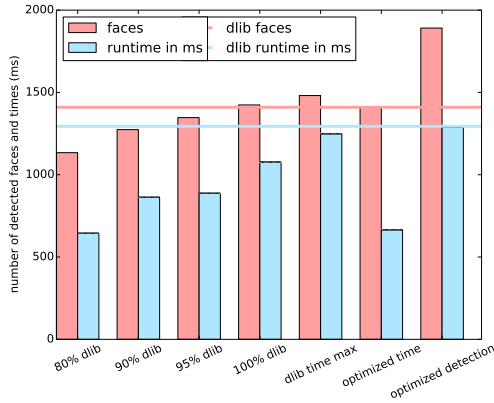


Figure 4.46: Best of all categories dlib vs SmartFace

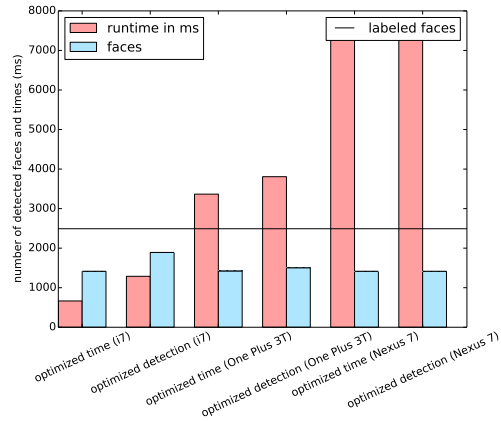


Figure 4.47: Direct comparison of devices

(Fig. 4.45b) images resulting in the best overall score, as indicated by Figure 4.45c.

Figure 4.46 shows a comparison of the best parameter combinations. The two horizontal lines are the baselines of *dlib* for runtime and number of detected faces without preprocessing, respectively. With an increased number of detected faces (80-100%), the runtime also increases for *dlib* with preprocessing. When reaching 100%, the runtime is still slightly better than *dlib* without preprocessing. Giving *dlib* with preprocessing the same amount of time as *dlib* without preprocessing results in a few more detected faces. On the other hand, *SmartFace* (i.e., the red/blue bars denoted by *optimized time* and *optimized detection*) is almost twice as fast as *dlib* with preprocessing, when limited to the same number of detected faces. If *SmartFace* gets the same amount of time as *dlib* with preprocessing, the number of detected faces increases significantly.

Device Performance

A comparison (see Fig. 4.47) of *SmartFace* running on all three devices shows that the high-end *OnePlus 3T* is about 2-4 times slower than the *i7 workstation*. The slightly outdated mid-range tablet *Nexus 7* is about twice as slow as the *OnePlus 3T*. All three devices yield about the same high face detection rate, but the runtimes range between 3 and 7 seconds for the mobile devices. For an emergency scenario with a DTN sharing photos, these runtimes are sufficiently fast.

The individual performance of the three devices is shown in Fig. 4.48. Across all

4 Disruption-tolerant Device-to-Device Emergency Communication

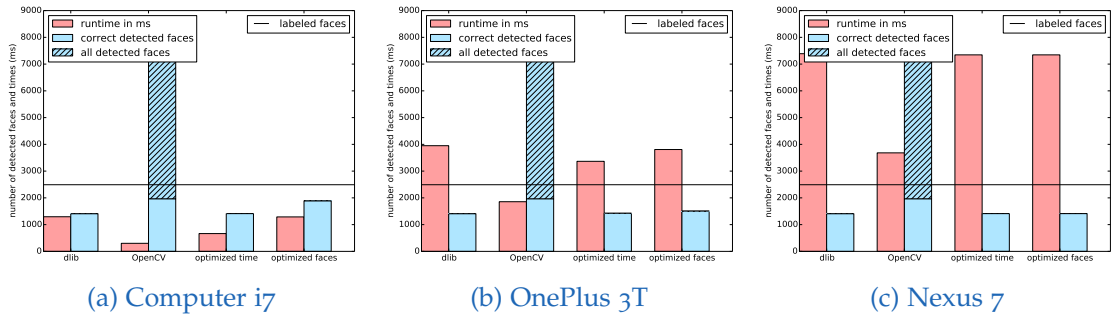


Figure 4.48: Individual device performance

Table 4.19: Transmission times for various link types

	100 Kb/s	2.1 Mb/s	54 Mb/s	150 Mb/s
2.8 GB	3,734 Min	177.78 Min	6.91 Min	2.49 Min
18 MB	24 Min	1.14 Min	0.044 Min	0.016 Min

devices, *OpenCV* is clearly the fastest algorithm, but with an unacceptably high false positive rate. Figure 4.48a shows that depending on whether *SmartFace* is optimized for time or detection rate, it clearly outperforms *dlib*. Even on the resource-constraint mobile devices, the number of detected faces is increased.

Storage and Bandwidth Savings

The complete full image test set is about 2.8 GB in size. When applying *SmartFace*, this leads to 18 MB of face images for the runtime-optimized variant and around 24 MB for the quality-optimized variant. Thus, *SmartFace* reduces the data roughly by a factor of 133 to 0.75% of its original size. This indicates how much storage space and network bandwidth can be saved when applying *SmartFace* prior to sharing data in a wireless on-demand hop-to-hop emergency network.

For illustration, Table 4.19 shows the transmission times of sending either the full 2.8 GB or the reduced 18 MB data set over different wireless link types: (a) 100 Kb/s (e.g., LoRaWAN), (b) 2.1 Mb/s (e.g., Bluetooth), (c) 54 Mb/s (e.g., WiFi 802.11g), (d) 150 Mb/s (e.g., WiFi 802.11n). Of course, the runtimes of *SmartFace* must be added to the transmission time at the first node. Since for a single image they range from 665 ms on the *i7* over 3.4 s on the *OnePlus 3T* up to 7.3 s on the *Nexus 7*, the runtimes for whole image test set are roughly 16 min, 84 min and 180 min, respectively. Therefore, there are different break-even points for the three devices. Fig. 4.49 shows that due to its computational power, the *i7* already pays off after 1-3 hops of transmissions for most link types; only for fast WiFi it takes about 8 hops to pay off, which is a quite low number in a scenario of DTN store-and-forward with epidemic routing. When looking an the slowest links such as LoraWAN and Bluetooth transmitting 2.8 GB of data is not realistic. Therefore, good compression or data reduction is a necessity. For both mobile devices, the break-even point for the faster WiFi connections is 12 hops or higher. Since these are hop-to-hop transmissions instead of classic TCP/IP connections and data will eventually be passed to every participant in the area, quite high hop-counts are likely

4.7 Applications for Disaster Response: SmartFace

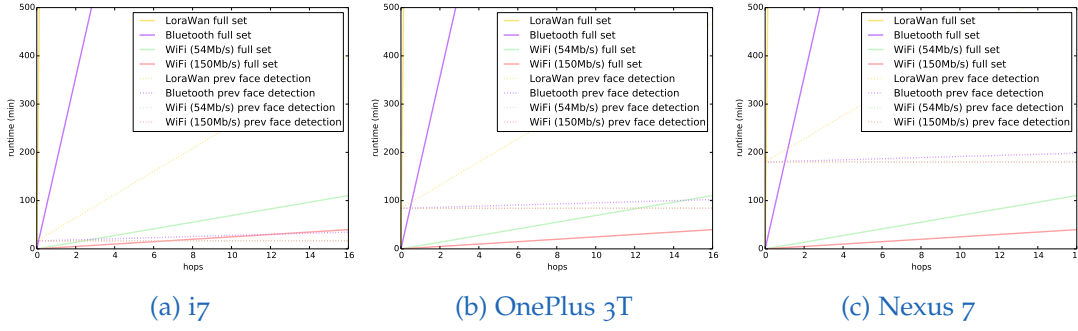


Figure 4.49: Comparison of transmission and optimization times on various devices.

in an emergency scenario. In reality, the impact of the runtimes on each mobile is much less, since no mobile device would input 3 GB of data at once, but each device would process its own images. A photo taken every now and then, processed directly and then shared epidemically does not have much influence on a single mobile device, but for the whole network, the consumed bandwidth and storage resources are influenced in a significant manner, as stated above.

4.7.6 Conclusion

SmartFace is a novel approach to perform face detection on mobile devices in an efficient manner for supporting the search for missing persons in wireless on-demand emergency networks. The approach relies on a two-stage combination of existing face detection algorithms, enhanced by region of interest selection, color space/depth reduction, resolution scaling, face size definition, image scaling, image cropping, and bounding box scaling. Experimental results have shown that the proposed approach improves both the overall face detection rate and the overall runtime compared to each of the individual face detection algorithms used alone, and also reduces the amount of data that needs to be stored on disk and sent over the network.

There are several areas for future work, such as (a) exploring the use of visual concept detection algorithms as filters to determine whether humans are present in an image; the most computation-intensive parts of *SmartFace* can then be skipped if no persons are present, (b) improving the runtimes of *SmartFace* on mobile devices by utilizing multiple CPU cores or GPUs, (c) applying *SmartFace* to non-scenario specific image sets (e.g., LFW or FDDB) for a general evaluation of our optimizations, and (d) combining *SmartFace*'s offline face detection capabilities with a decentralized on-device face recognition approach to find missing persons or family members by just participating in a DTN and relaying data.

4.8 Applications for Disaster Response: UV4EC

4.8.1 Introduction

Rescue operations during and after disasters often expose rescue teams to high risks. Therefore, more and more unmanned vehicles (UVs) are used on the ground, in the air or in the water to support rescue operations. Typically, such UVs operate either in a semi-autonomous manner or are completely controlled by remote human operators. For example, in the ruins of Fukushima Daiichi remotely controlled robots were sent to highly contaminated areas. Remote control requires a reliable communication infrastructure to coordinate UVs and to increase their operation radius significantly. However, during the first hours of a disaster event, the existing communication infrastructure might be severely damaged, disrupted or overloaded due to network congestion. Thus, re-establishing basic ways of communication during a disaster despite fragmented IP networks and totally or temporarily disrupted network links is a key step in successful disaster management and rescue operations.

An emergency communication system should not only support UVs, but also human rescuers and civilians who are still in the disaster area. Since connectivity cannot be easily established in the entire affected area, it is more likely that small islands of devices connected to each other will evolve, with limited bridges between these islands. These islands can be formed by team members operating in the field, people trapped in houses or waiting in temporary shelters, and clusters of cooperating UVs. By using store-carry-forward technologies, humans or UVs can act as carrier pigeons and deliver data between islands, thus spreading information in an epidemic fashion. This approach can be used for person-to-person communication, in a way similar to SMS and various messengers, or for sensor data such as images from smartphones or temperature and Geiger counter readings from UVs that are shared with the public. Gathering this information and processing it in the rescuers' operations and control center is helpful for situation analysis and coordination of rescue endeavors. In general, long distance, real-time unicast communication is not possible in such a scenario. The chances of establishing a successful multi-hop connection to a specific node are increased by altering the objectives of UVs, such as drones, to incorporate air bridges as part of their mission. In this case, a secure mesh routing algorithm is needed for real-time communication in the emergency communication system.

Here, a novel emergency communication system that relies on delay-/disruption-tolerant networking (DTN) for non-time critical tasks and direct mesh connections for prioritized tasks that need real-time feedback is presented. It is used for the distribution of sensor data, human-to-human communication, as well as direct and indirect control of unmanned ground vehicles (UGV) and unmanned aerial vehicles (UAV). To demonstrate the real-world applicability of the developed emergency communication system, called *UV₄EC* (Unmanned Vehicles for Emergency Communication), it is run on *RoboCup Rescue* proven [114] robots as well commodity mobile devices, and various drones, and support our operations and control center software for disaster management. Experimental results indicate that a combined DTN and mesh networking approach is a good compromise for providing emergency communication based on semi-automatic and remotely controlled autonomous UVs. In particular, the following contributions are made:

- A novel hybrid DTN and mesh networking communication middleware for disaster scenarios.
- A novel approach to emergency communication where a semi-autonomous UGV establishes UAV-based communication bridges between dynamically forming communication islands of mobile devices.
- An operations and control center for commanders to get a more complete situation overview of the disaster scenario and to control the rescue mission by sending high-level commands to mobile rescuers or UVs.

Parts of this section have been published in [7].

4.8.2 Related Work

Several approaches that address emergency communications needs have been presented in the literature [49]. Some of them require special hardware (e.g., radio-link technologies or satellites), are engineered for specific tasks, or are not usable on consumer-grade hardware with commodity operating systems [50], [52], [53], [55], [60], [115]–[117]. Since our focus is to enable as many people as possible to communicate in an emergency network, our approach attempts to leverage commodity hardware and software, such as WiFi-enabled mobile end-user devices, where possible.

Other research has shown that DTNs and mesh networks are viable solutions in emergency scenarios [51], [56], [118], [119]. By leveraging ubiquitous mobile devices, such as smartphones or tablets, novel communication solutions have been proposed [57], [59], [120]–[122] and evaluated [123]. Since both DTNs and mesh networks have distinct capabilities, we incorporate a dual networking stack in *UV4EC*, which uses mesh networking for real-time communication and DTN for data sharing and messaging.

While the use of both UAV and UGV systems (and their combination) for search and rescue applications has received considerable attention from the research community [124], the combination of robots with both DTN and mesh networking has not been studied extensively. However, based on the experiences with real-world deployments [125], [126] and the associated communication difficulties, this appears to be a promising direction of research. While mesh-based approaches have been used successfully with UVs before [127], the body of research on the additional use of DTN with UVs is limited to mainly theoretical results [128]–[130].

Collecting, analyzing, and visualizing emergency data at a central point (e.g., in a control center) to provide a situation overview for rescuers is crucial in disaster management [131]. However, many proposed control centers are stationary and rely on centralized approaches, assuming Internet connectivity to exchange data or to control UGVs [132]. Only a few control centers have been designed for mobile use and decentralized communication [133], [134]. For instance, the SENEKA project presents an adaptable and scalable ground control station integrated in a van for gathering sensor data from stationary sensors, mobile ad hoc networks and mobile sensor platforms [134]. In addition to the data channel, it establishes a control channel to control UVs in the field. Inspired by this work, here a lightweight offline control center is presented, running on a customary laptop for mobile use and relying on mesh networking and DTN for decentralized communication while providing a similar range of functions as state-of-the-art approaches, namely data gathering, fusion and processing, providing visual

4 Disruption-tolerant Device-to-Device Emergency Communication

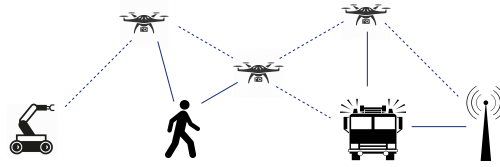


Figure 4.50: Emergency communication scenario

situation overviews, as well as realtime and delay-tolerant control of semi-autonomous UAVs and UGVs.

4.8.3 UV4EC's Design and Implementation

The design of *UV4EC* supports unmanned vehicles and professional responders as well as civilians in the affected area. All communication links can be volatile and connectivity is expected to be lost. Given an opportunity to exchange data, each peer will communicate with all of its neighbors for maximum data distribution, as indicated in 4.50.

Unmanned Ground Vehicles

The UGVs used in *UV4EC* employ ROS as a robotics middleware. ROS has become the de-facto standard in the robotics community in recent years. Based on a modular software design, components for different tasks like navigation and perception can be exchanged and adjusted for use with different vehicle platforms. For *UV4EC*, we use a tracked mobile robot based on the Taurob Tracker platform. To support multimodal sensing in disaster environments, the robot is equipped with a comprehensive sensor suite consisting of (among others) a spinning 3D LIDAR, a thermal camera, a 30x optical zoom camera, and a depth camera. The camera sensors are mounted on a sensor arm to provide flexibility in sensor positioning. The robot can perform autonomous exploration of unknown environments, using a Simultaneous Localization and Mapping (SLAM) approach to learn the environment and map and simultaneously localize objects within the environment [135]. The robot operator can select between teleoperation or autonomous operation at any time.

Unmanned Aerial Vehicles

Since the focus is on the communication aspects and to avoid the (administrative and organizational) overhead of deploying real UAVs, multiple simulated UAVs based on the Gazebo simulator are used. It allows us to simulate multiple UAVs with dynamics. The UAVs establish communication bridges between UGVs and the operations and command center (OCC) from time to time, i.e., the UAVs should position themselves near specific prescribed locations between UGVs and the OCC to enable communication. The task of allocating and getting to these positions is solved cooperatively and decentrally; necessary control inputs for each UAV are computed locally on board of the UAVs by applying a feedback and optimization-based vehicle controller [136], [137]. This vehicle controller uses a mixed logical dynamical (MLD) approach to model the multi-vehicle system [138] consisting of every UAV within the communication radius.

Based on the MLD formulation, an optimal control problem is set up, in which the number of and distance to unoccupied bridge locations is minimized subject to vehicle dynamics, allocation logic and constraints regarding collision avoidance. This optimal control problem is solved over a limited time horizon in a model predictive control fashion so that a mixed integer linear program has to be solved in every time step. The first elements of the resulting control input sequence are applied by each individual UAV.

Emergency Communication

In disaster scenarios, a number of communication services and applications are required to ensure effective disaster response. These applications include human-to-human (message-based) communication, sensor data sharing for situational reporting, and also “real-time” control channels for operating UGVs and UAVs from a remote location. Unfortunately, during a disaster, the local communication infrastructure, such as cell towers and Internet-connected WiFi hotspots that would normally be used to support these applications, might be unavailable either due to poor coverage at the disaster site or because it has been destroyed by the disaster.

Thus, the design decision to rely on decentralized ad-hoc communication technologies to provide connectivity in such extreme environments. To enable a wide range of applications with different communication requirements, a dual networking stack consisting of (i) a mobile cloud based on delay/disruption-tolerant networking that can be used for messaging and information sharing applications, and (ii) a highly adaptive end-to-end communication protocol based on wireless multi-hop routing that quickly finds and exploits communication bridges and can be used for control applications with “real-time” feedback is employed. The two components are presented in detail below.

Mobile Cloud

In a network consisting of many mobile and fast-moving nodes including UGVs and UAVs, the contact times between two nodes are typically short. When communication links are only available for short periods and in the near vicinity, classical multi-hop routed network communication is hardly possible. As an alternative, store-carry-forward based DTN has been proposed especially for challenged environments such as space communication and emergency communication. In DTN, nodes are considered as “data mules”: they carry their own messages as well as messages from others. When two nodes meet, they exchange all messages they are carrying by replicating them. Using this approach, messages are spread in an epidemic fashion throughout the network, which increases the chances that a message eventually arrives at its destination. In DTN, delivery performance highly depends on node mobility. This makes it attractive for a disaster scenarios where mobile devices of civilians and professional disaster response staff as well as UGVs and UAVs are present and physically move around. Due to its best-effort service, DTN is suitable for applications such as text messaging, collecting sensor data, and transmitting geo-location updates.

This research relies on the Serval Project⁷⁴ to realize a DTN protocol for *UV4EC*.

⁷⁴<http://www.servalproject.org>

4 Disruption-tolerant Device-to-Device Emergency Communication

Serval provides basic messaging with built-in end-to-end encryption and a portable C code base for further extensions. To verify the viability of Serval in our disaster scenario, an in-depth performance analysis of the existing protocol and software was conducted in Section 4.3. Based on the results, the protocol overhead was reduced by dynamically adjusting the neighbor announcement interval (Sec. 4.4) and transparent computation task offloading was implemented to preserve local resources (Sec. 4.5). To allow the integration of Serval in *UV4EC*, pre- and post-receive message filters as well as content hooks that are triggered when manipulating the data store^{75,76} were developed. Furthermore, more advanced features such as append-only sensor logs and file updates for rapid prototyping and deployment in the field are supported.⁷⁷ For non-mobile users, custom ways of interfacing with the Serval application are currently in development, including a web interface⁷⁸, standalone desktop application⁷⁹, and a full-screen console application⁸⁰. The final system enables any WiFi-enabled UNIX-based system, such as smartphones running Linux, Android, or MacOS, to communicate with each other. This includes one-to-one encrypted text messaging and file transfers, and publicly shared information such as images or position logs. It can easily be ported and used by new specialized systems such as rescue robots, drones or static sensor nodes in addition to the ones we are already using.

Communication Bridges

Certain applications, such as directly controlling a UGV or UAV, require real-time communication. This cannot be realized using the mobile cloud, since it does not provide delivery guarantees or feedback. Therefore, control messages might never or only very lately reach their destinations, which would render UGV/UAV control unreliable. To solve this problem, it is resorted to classical store-and-forward routing where messages are forwarded from one node to the next node until it reaches its destination. If a destination is not reachable, for example, because the network is partitioned or the node has moved out of range, messages should be dropped: in this case, there is no need to buffer packets, since one is not interested in late deliveries. However, the system should be able to notify the user in a timely manner that communication was unsuccessful or that an ongoing communication was interrupted.

For *UV4EC*, SEMUD[139] is used that provides an end-to-end communication protocol is employed. It supports on-demand route discovery and constantly adjusts its active routing paths. This is important for a highly mobile network where topology changes are frequent. In other words, SEMUD maintains a stable route while there is active communication between two nodes, for example, between the operator's device and a UGV; and does not generate any traffic while end-to-end communication is inactive. This is in contrast to classical proactive routing protocols for mesh networks such as OLSR⁸¹. SEMUD achieves these goals by a combination of per-packet feedback, the integration of routing with actual data transmission, the use of reliability as a distance

⁷⁵<https://github.com/umr-ds/serval-contentfilters>

⁷⁶<https://github.com/umr-ds/serval-dna/tree/nicer-filters>

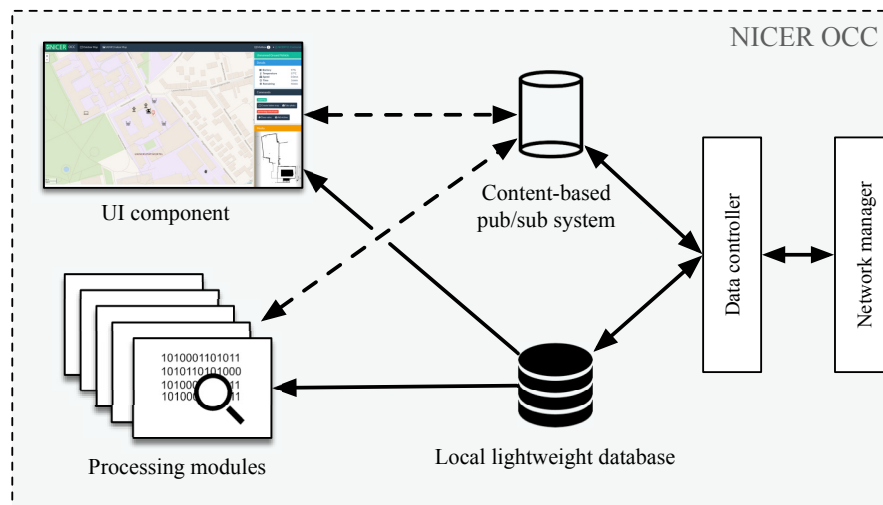
⁷⁷<https://github.com/gh0st42/servalshellscripts>

⁷⁸<https://github.com/umr-ds/serval-web>

⁷⁹<https://github.com/gh0st42/ServalDesktopApp>

⁸⁰<https://github.com/gh0st42/sdnatui>

⁸¹<https://tools.ietf.org/html/rfc7181>

Figure 4.51: Architecture of *NICER OCC*

metric, and lightweight cryptographic mechanisms. These features allow SEMUD to be highly adaptive to topology changes and resilient to a wide range of common attacks on wireless routing protocols even in the case that a device is compromised or captured by an adversary. Furthermore, a prototypical and portable C++ implementation of the protocol is freely available⁸².

Operations and Control Center

To provide a more complete situation overview and coordinate rescue endeavors, a user-friendly operations and control center (termed *NICER OCC*) was developed, that is commonly located outside of the affected region. It comprises (1) *data gathering, fusion and processing*, (2) *visualization*, and (3) *coordination* of mobile rescuers or semi-autonomous unmanned vehicles (UVs) with specific focus on high-level commands, namely *exploring* (e.g., mapping, taking photos) or *performing critical tasks* (e.g., closing a valve). To show the applicability for the outlined disaster scenario, a proof-of-concept prototype that relies on the previously described communication network and can run on a customary laptop was implemented.

Data Gathering, Fusion and Processing The *NICER OCC* strives to collect a comprehensive picture about the situation and the interactions of mobile rescuers and UVs in the field. Pursuing a loosely coupled and event-driven architecture (cf. Fig. 4.51), the system or the *data controller* first gathers and fuses sensor data of multiple nodes from the decentralized ad-hoc communication network; and *second*, (a) persistently stores them in a *local lightweight database* for historical views, as well as (b) publishes them to a *content-based pub/sub system* for push updates to the subscribed modules, such as *processing modules* or the *UI component*. Subscribed processing modules can then asynchronously analyze the data (e.g., image editing, object recognition) and publish the results back to the pub/sub system, where other subscribers (e.g., the UI component) can use them. The computational workload (i.e., processing modules) can also be

⁸²<https://seemoo.de/semud>

4 Disruption-tolerant Device-to-Device Emergency Communication

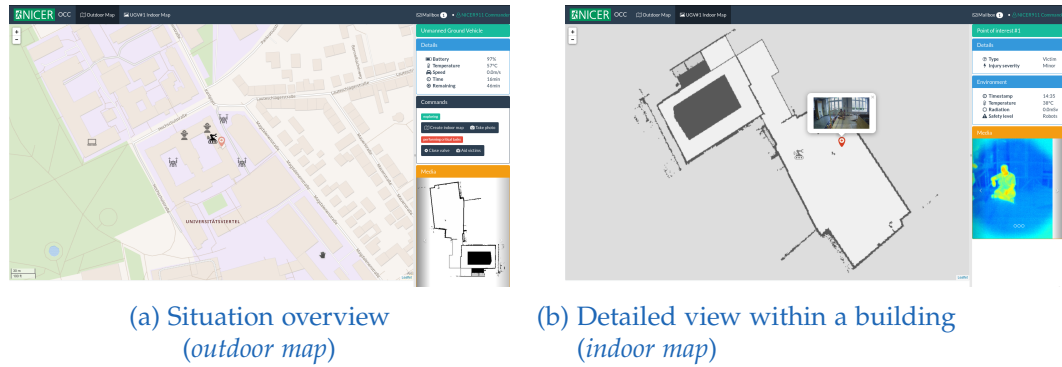


Figure 4.52: UI component of NICER OCC

moved to the network by integrating in-network processing approaches [140], which is especially designed for such infrastructure-less scenarios. In urban environments, the system can also utilize upgraded home routers as DTN communication bridges and computing nodes [141].

Visualization The *UI component* of the system provides a visual overview of the situation by displaying the collected and processed sensor data (cf. Fig. 4.52). Modern lightweight web technologies are used to accelerate UI development, to provide an easy to learn and use application, as well as to easily distribute and port the system (*cross-platform compatibility*) in infrastructure-less disaster scenarios.

Figure 4.52a shows the offline (*outdoor*) map, which is supposed to provide an entire situation overview for the commanders. It displays and updates all *units* in the field (e.g., UVs, mobile rescuers) who are at least equipped with communication technology and location sensor, *points of interest* (POIs), and overlaid *indoor maps*. By clicking on a unit, a detail view on the right side of the map appears, representing all relevant collected information (e.g., health status) and possible high-level commands (e.g., take a photo). The same is true for a POI, which geographically marks a situation that the commanders may find interesting, e.g., detected victims or damages. Receiving new data (e.g., status or location updates) by listening on the pub/sub system, the UI component only updates the visual representation of the referenced unit or object without refreshing the whole map.

To provide a more detailed view for inaccessible buildings (e.g., a nuclear reactor) to the commanders, the NICER OCC can display and update indoor maps created via laser scanning by on-site UGVs (cf. Fig. 4.52b). The indoor map view is based on the user interface of the outdoor map to apply same interaction concepts.

Coordination The NICER OCC also enables commanders to coordinate the units in the field by sending high-level commands. Two types of commands to control semi-autonomous UAVs and UGVs are defined, namely (i) *exploring* (e.g., mapping, taking photos), and (ii) *performing critical tasks* (e.g., closing a valve; handing special equipment, emergency rations or medical kits over to injured people). Internally, these commands are encoded and sent via messages over the available network. For a high-priority command or a direct control of an UV, the NICER OCC can easily request a so-called *communication bridge* from the units in the field (cf. Section 4.8.3). The NICER OCC

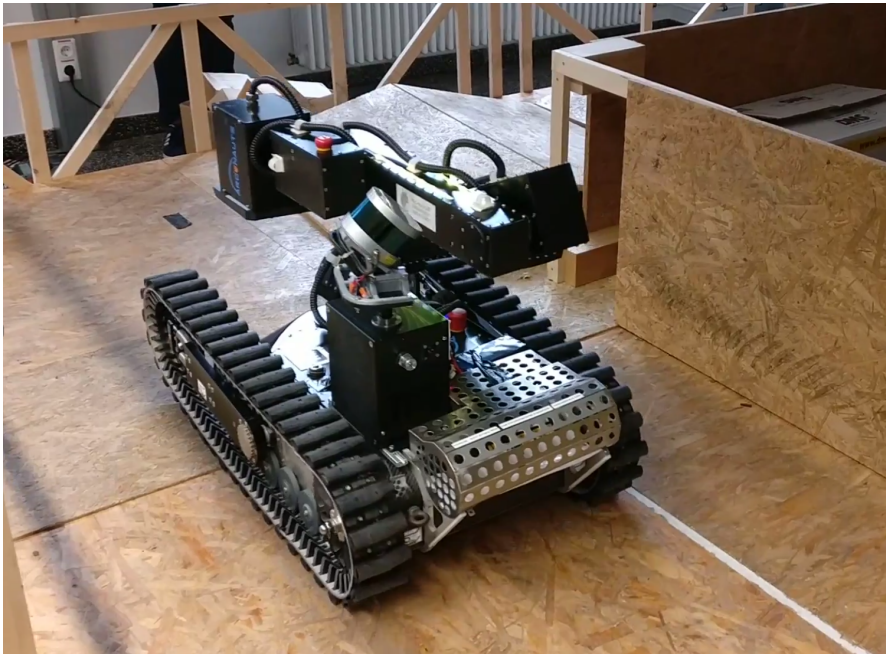


Figure 4.53: Hector Tracker robot operating in a simulated disaster scenario.

can also be extended by integrating civilians who use our mobile application for smartphones (cf. [142]).

4.8.4 Experimental Evaluation

UV4EC is evaluated by performing real-world tests (4.8.4) and performing comprehensive simulation and emulation of various nodes in realistic setups (4.8.4) based on the generated data.

Real World Setup

To provide a test scenario that can be transferred to real applications, real-world disaster scenarios is emulated by relying on well-established approaches for evaluating disaster response robots. This includes NIST standard test methods [143] that are used in the RoboCup Rescue competition. They are designed to provide reproducible test setups representative for the challenges encountered in real disasters. Figure 4.53 shows an image of the robot operating in the scenario.

To simulate the use of a robot for victim search in a disaster scenario, multiple persons are placed in a simulated disaster zone, simulating trapped victims. The Hector Tracker vehicle is then used to explore the environment and search for victims. The robot can be controlled by teleoperation or can operate fully autonomously, and the used control paradigm can be changed at any time. Using onboard sensors, it generates a map of the environment including geometric (point cloud) data and found victims or objects of interest.

Communication requirements for robot operation depend on the operation mode. For fully autonomous control, neither up- or downlinks are required, but often desirable to monitor the vehicle. For pure teleoperation, connectivity between robot and operator

4 Disruption-tolerant Device-to-Device Emergency Communication

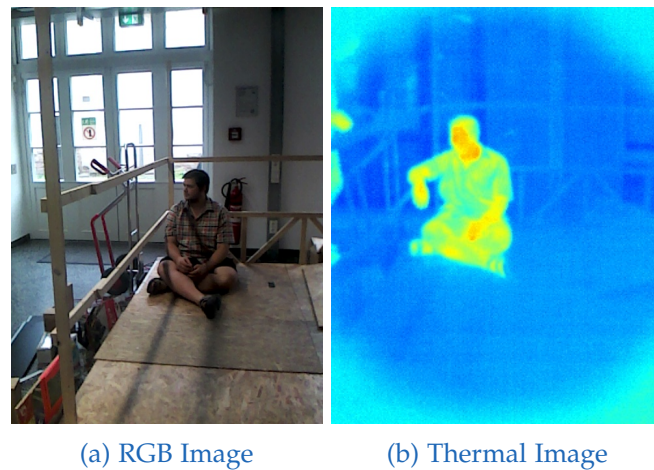


Figure 4.54: Robot sensor data for a "victim found" event

station must be established. The downlink (robot to operator) direction then requires the transmission of low latency image data.

Independent of the operation mode, the robot's mission state needs to be communicated as the main deliverable for responders. It mainly consists of robot pose, 3D environment map as well as event-based tracking of objects of interest, such as finding a trapped victim. Since all related computation tasks take place onboard the robot, all mission state related information can be communicated in a delay-tolerant manner. This motivates the use of image data for evaluating communication performance in Section 4.8.4. Figure 4.54 shows example camera imagery that is associated with a "victim found" event onboard the robot when a victim has been automatically detected.

Lab Test Environment

To get a basic understanding of the performance of *UV4EC*, several experiments in a controlled simulated and emulated environment derived from scenarios faced in Section 4.8.4 were performed. The following results can be directly transferred to real world deployments, due to the focus on emulation of systems instead of simulated algorithms.

Test Environment All tests were performed on an i7-4771 CPU @ 3.50GHz, supporting 8 threads, with 32 GB of RAM. The nodes were simulated using a combination of the Common Open Research Emulator (CORE)⁸³, Gazebo⁸⁴ robot simulator and ROS⁸⁵. The system was designed with a static operations and command center node *center1*, a robot slowly moving in one direction until it gets stuck at a building entrance *robot1* and three highly mobile drones *drone1-3* with various objectives. These drones either have fixed points of interest for their mission or circle around specific positions. Furthermore, they periodically seek contact to *center1*, and *robot1*. The setup is shown in Figure 4.55 where an accident in the chemistry building of the university of Marburg is simulated. In the

⁸³<https://www.nrl.navy.mil/itd/ncs/products/core>

⁸⁴<http://gazebo.org/>

⁸⁵<http://www.ros.org/>

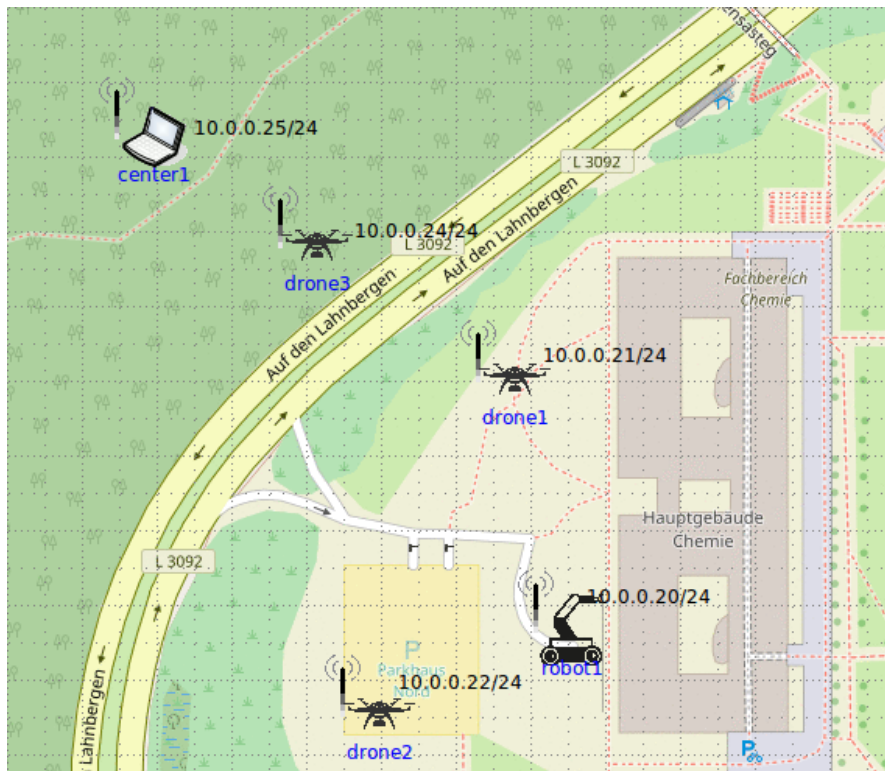


Figure 4.55: Simulation setup with three drones, one ground robot and a command center operating on our university campus

simulations, the area covered by all involved entities is about 250.000 m^2 . Each node has a 802.11g WiFi interface in ad-hoc mode for direct mesh communication. Therefore, the bandwidth cannot exceed 54 Mbit/s and has a maximum simulated range of 80 meters. Each experiment was repeated 10 times and ran for about 260 seconds.

Communication Opportunities For delay-tolerant communication, it is important how often peers get a chance to exchange their data with others. In the simulation, it was determined, using one second intervals, how many peers are currently in communication range.

Overall, there were over 800 total contact opportunities in the entire simulation across all nodes. The individual number of contacts per node is shown in Figure 4.56. The number of opportunities over the runtime per node is depicted in Figure 4.57. It shows that within this area of action and without any specific targets for the autonomous drones regarding communication, there are plenty of opportunities to exchange data, even though the mobile drones have more opportunities than the rather static or remote nodes, such as *center1* and *robot1*.

Delivery Times Knowing that many opportunities exist for data exchange in this scenario, one must still determine how fast responses to individual commands can be expected in such circumstances.

Two different tests were performed to analyze the delivery times. The first test injects image data at *robot1* and measures how long it takes for this data to arrive in

4 Disruption-tolerant Device-to-Device Emergency Communication

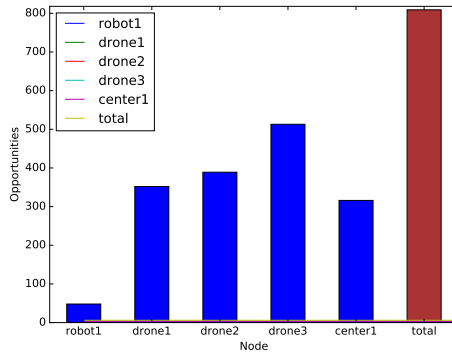


Figure 4.56: Opportunities for data exchange

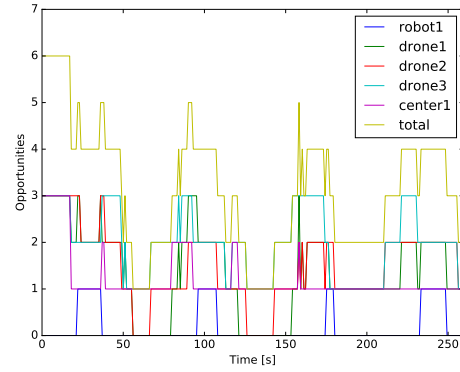


Figure 4.57: Opportunities for data exchange over time

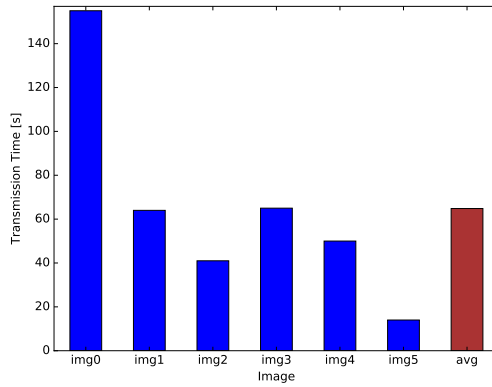


Figure 4.58: File distribution times

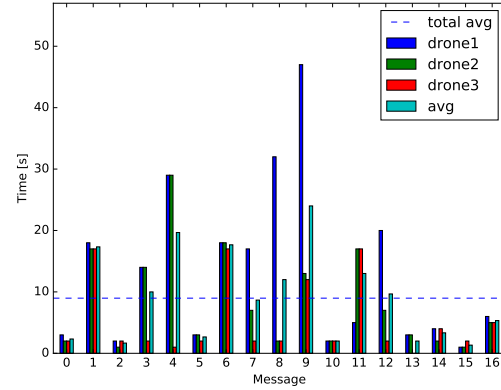


Figure 4.59: Message distribution times

the operations and control center (*center1*). In the second test, *center1* wants a direct connection to *robot1* and therefore sends the commands for an air-bridge to all three drones. It is measured how long it takes for all drones to receive the command. Both experiments were repeated with different starting times and therefore different geo-spatial distributions of nodes.

File distribution depends on the file size. In the experiments, 1080p images were sent as they are recorded by the physical robot. The transmission times are shown in Figure 4.58. Average file transmission time is about 60 seconds with the worst time of about 155 seconds. Without this image transmission, the average time goes down to about 46 seconds. This can be explained with bad timing, no peers in range or moving out of range during transmission, and Serval needing some time to recover from failed transmissions. The transmission always needs at least 2-3 hops to travel from *robot1* to *center1*, considering that the area of action covered by an average transmission time of 60 seconds is more than acceptable.

When it comes to message delivery to the drones, the picture is similar. If messages are sent faster than they can be delivered, they are transmitted together, arriving at the same time even although they have been sent at different points in time. The arrival times are shown in Figure 4.59. Since text messages are much smaller than images, the chances of a successful transmission even with only short contact periods is much higher, resulting in an average transmission time of about 9 seconds in this

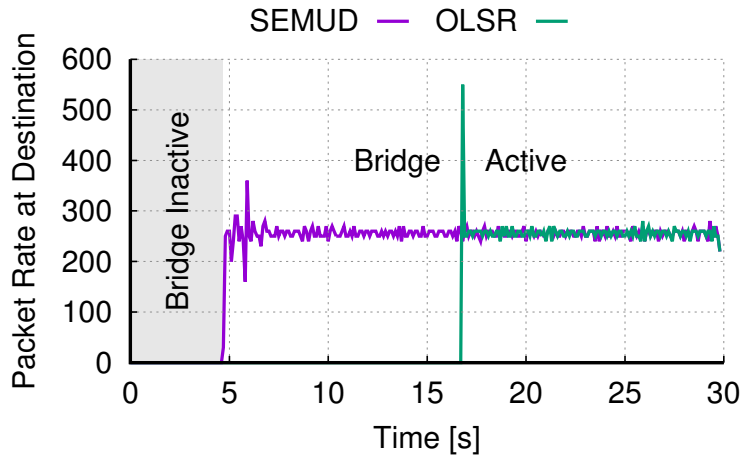


Figure 4.60: Reaction to a newly established communication bridge of SEMUD and OLSR

scenario. Considering the worst delivery rates recorded in the experiments, a message still reaches its destination in under one minute. Thus, even with the rather low number of participants in the simulation, direct control can be triggered within a quite short period of time.

All these numbers are highly scenario specific, and by having humans with network devices also in the affected area, even better delivery rates can be achieved. Having the drones programmed with multiple objectives, one seeking opportunities for data exchange, gives a powerful tool to dynamically adapt to the environment and the tasks at hand.

Connection Establishment The responsiveness of the communication system to newly established communication bridges, i. e., when drones have moved into positions to allow a multi-hop connection from the control center to a robot, was also evaluated. For the experiment, *center1* generates UDP/IP traffic to *robot1* using the iPerf⁸⁶ tool. The incoming packets are recorded at *robot1* over time using *tcpdump* and their arrival times can be seen plotted in Figure 4.60. The figure also marks the point in time when the drones have aligned and the communication bridge has been established.

One can see that SEMUD almost instantaneously is able to deliver messages to the destination after the communication bridge has become active. This is due to SEMUD's design of integrated route discovery and data transmission: there is no explicit protocol that needs to be run prior to actual data transmission, but data is piggybacked to every message that is sent, which keeps delay at a minimum. In addition, SEMUD essentially follows a trial-and-error approach: every message is flooded through the network until feedback from the destination is received, in which case SEMUD switches to unicast transmissions to reduce network load. In contrast, OLSR is significantly slower in exploiting the communication bridge: it needs approximately 17 seconds to find a path. This is due to the fixed interval at which OLSR exchanges routing information. Note that this additional delay is not a one-time cost, but has to be paid whenever the topology changes. This is especially problematic when long-term continuous control of a moving robot is required and more drones are added to maintain the bridge.

⁸⁶<https://iperf.fr>

4 Disruption-tolerant Device-to-Device Emergency Communication

Message Processing Besides the delivery time of messages through the network, it is also important that the system has low message processing times at the network devices to ensure fast responses. Therefore, the NICER OCC is evaluated by system performance tests with respect to the entire message life-cycle between the network and the UI, i.e., from the arrival of the message at the OCC through to the visualization of the contained sensor data in the UI or on the map, so that the commanders can perceive them.

Both directions of message processing are considered, (a) reading *sensor data* from the network (including deserialization) and updating the UI, as well as (b) sending *commands* from the UI to the network (including serialization). As performance measures, *latency* and *throughput* are used. Latency is the time required to process one message. Throughput is the number of such messages processed per unit of time. It is important to note that one message can contain multiple sensor data such as location updates. However, to analyze the different message types and make the tests comparable, the number of sensor data or commands is limited to one per message for the lab test.

Table 4.20: Performance test of OCC w.r.t message processing

Message type	Latency [ms]	Throughput [1/s]
Sensor data (location)	26.1 ± 1.3	53.8 ± 4.0
Sensor data (image)	76.6 ± 2.5	12.5 ± 0.5
Commands	69.1 ± 1.8	19.6 ± 0.3

Table 4.20 shows the results of the performance tests. One can observe that messages with different kinds of sensor data are processed differently: a message with location data needs about 26 ms to be processed in the OCC while messages with image content need roughly three times more (~ 76 ms). This is also reflected in the throughput measurements. Considering the other direction, commands can be sent in about 70 ms from the UI to the network, which is more than sufficient to be able to smoothly control the rescue mission.

All in all, the message processing of the operations and control center is negligible compared to the delivery times of the messages through the network. Thus, there is no lack of performance for the OCC; quite the contrary, the OCC supports situation overviews with high-frequency updates in the magnitude of the human eye or screen refresh rate.

4.8.5 Conclusion

In this section, *UV4EC* was presented, a novel emergency communication system involving unmanned vehicles. By combining a DTN-based mobile cloud infrastructure for non-time-critical tasks with reactive mesh routing for real-time interaction, novel ways of operating UGVs and UAVs are provided that also support humans in the affected area. UGVs and UAVs are used to bridge communication gaps that otherwise would significantly reduce their radius of operation. Also a lightweight operations and control center that complements *UV4EC* to provide all relevant functionalities (i.e., data gathering and processing, providing visual situation overviews, and sending high-level commands) to commanders for coordinating rescue missions was developed.

Furthermore, the viability of the approach by experimental evaluation and real world deployment was demonstrated.

There are several areas of future work. For example, additional work should be invested in developing further DTN-aware control mechanisms for various robot operations. Furthermore, the coordination of the UAVs can be fine tuned and optimized for specific scenarios. Depending on the tasks at hand, data prioritization should be considered at the UAV/UGV, DTN and/or routing levels for improved network performance. Finally, for low-priority tasks (e.g., analyzing local sensor data), it is planned to move the computational workload from the control center to the network, i.e., the data should already be analyzed by the nodes in the network.

4.9 SEDCOS - Secure Disaster Communication

4.9.1 Introduction

Communication technologies are integral to disaster relief operations. The solutions presented in the previous sections leverage the ad hoc and disruption-tolerant networking (DTN) capabilities of mobile devices to create opportunistic communication networks. In DTNs, all devices store, carry, and forward data to form a dynamic, infrastructure-less, and self-organized network. Coverage is increased by adding more devices to the network. In particular, the approach can be applied to mobile commodity devices, such as smartphones, tablets, and laptops, which are ubiquitous and provide diverse ad hoc communication capabilities (e.g., Wi-Fi and Bluetooth). In this way, people can continue using their personal devices to request or offer aid, obtain information from emergency services, or contact relatives and friends.

However, such opportunistic networks are susceptible to a wide range of security attacks due to their wireless, cooperative, decentralized, and resource-constrained nature. For instance, during wars or terror attacks, adversaries may subvert the communication system to disrupt disaster relief operations by injecting false information or performing denial-of-service (DoS) attacks. Furthermore, panicked people may spam the network with messages, unintentionally jeopardize availability.

Thus, a practical emergency communication system must ensure confidentiality, authenticity, integrity, and availability, but these properties are difficult to achieve during adverse events. Existing proposals either lack disaster functionality or provide an insufficient level of security [144]–[147]. High data availability and reliability are crucial for emergency notifications and distress signals. Prior work has improved reliability, but has not assessed secure prioritization mechanisms that work reliably under attack.

In this section, *SEDCOS*, a secure device-to-device communication system for disaster scenarios, is presented. The main contributions are:

- a secure communication substrate with message prioritization and a management scheme that delivers messages reliably and is resilient against flooding DoS attacks, and
- large-scale network simulations showing *SEDCOS*'s effectiveness in maintaining high delivery rates under attack and revoking user certificates in the field.

Parts of this section have been published in [8].

4.9.2 Related Work

Typical security targets in opportunistic networks are authentication and integrity of messages [148], secure routing [149], and confidential as well as anonymous end-to-end communication [150]. Identity-based Cryptography (IBC) is a frequently suggested solution, since traditional public key cryptography is often regarded as unsuitable for opportunistic networks due to the need of accessing public keys, certificates, and revocation information from central online servers [151]. To eliminate central authorities, fully decentralized trust-based concepts [152], [153], or approaches based on threshold-cryptography [154] have been proposed. However, existing works do not address the unique challenges of disaster relief communication, such as a high delivery

rate (emergency messages), immobility of individual users (trapped victims), role-based authentication, or insider attackers. Denial-of-service attacks on unauthenticated DTNs have been evaluated, but contrary to previous findings [149], we show that authentication is essential for reliable operation. Other works have attempted to hinder flooding attacks by setting explicit rate limits and trying to detect misbehaving nodes using a complex distributed detection mechanism [155].

4.9.3 System Model

Store, Carry, and Forward

Instead of relying on infrastructure, DTN-enabled devices exchange messages directly using WiFi and Bluetooth. DTNs exploit user mobility to increase coverage: devices act as “data mules” that store their messages as well as messages from other users, carry them, and finally forward them to the destination upon contact. This way, messages propagate in an *epidemic* manner from device to device until they reach their destinations. DTN performance is typically worse than that of infrastructure networks but is preferable to no communication at all. Nevertheless, devices with Internet connectivity (cellular or WiFi access points) can opportunistically act as “wormholes” used for rapid message distribution to isolated parts of the network.

Communication Model

Communication in an emergency scenario is either *one-to-one* (contact with friends or family), *many-to-many* (within task forces or departments), or *one-to-many* (emergency notification broadcasts). Due to the inherent delay of DTN-based communication, only small messages, such as text and prioritized distress messages (including additional information, such as GPS location of the sender), serving a similar purpose as the classic “112” emergency call, are allowed in SEDCOS. Compared to rich media (images, voice, video), information in text messages is more compact, thus, uses the limited resources of DTNs more efficiently.

Adversary Model

An adversary *Adv* is considered who can mount network attacks and compromise network entities. Specifically, *Adv* can eavesdrop, manipulate, forge, or drop messages. Furthermore, *Adv* can assume a limited number of entities, either by compromising or stealing devices or by registering multiple times in our system. Unlike the classic Dolev-Yao adversary model, *Adv* controls only a part of the communication channel and a portion of all network entities. Moreover, *Adv* cannot break cryptographic primitives or tamper with the *root authority* (see 4.9.4).

4.9.4 Secure Key Management

Establishing trust is important to satisfy the security requirements. For this purpose, a centralized trust model using a Public Key Infrastructure (PKI) is employed, as shown in 4.61. The PKI consists of multiple hierarchically organized certificate authorities (CAs), whose root is a dedicated authority named *root authority* (RA). The RA serves as a trust anchor, maintains the emergency communication software, and distributes

4 Disruption-tolerant Device-to-Device Emergency Communication

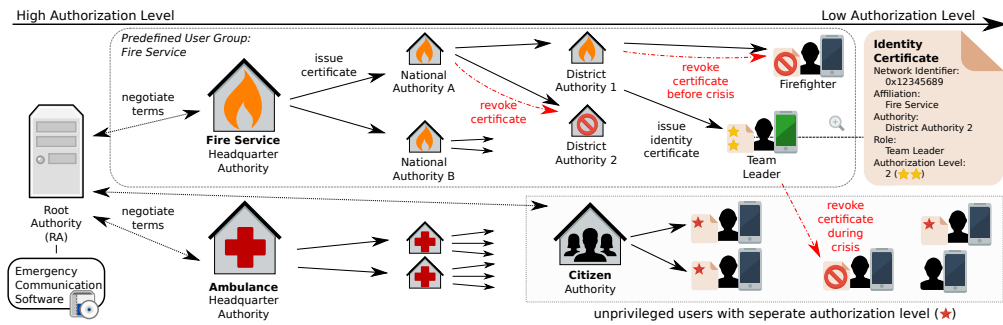


Figure 4.61: Illustration of our secure key management. The authorization level decreases from left to right, except for citizens.

the software if infrastructure access is still available. In an initialization phase, before the actual crisis, the RA establishes relationships to organizations or governments that want to participate as *authorities* in the emergency communication system. All authorities initially undergo a rigorous audit by the RA, since their authenticity and trustworthiness are crucial to the overall security. As part of the audit process, RA and authority agree on *user roles* as well as preconfigured *user groups* that the authority introduces to the network. For instance, in 4.61, the fire service organization added the user roles *team leader* and *firefighter*, and arranged a preconfigured user group *fire service*, so users can particularly address all firefighters when sending a message. Authorities manage their own PKI and, hence, maintain one or multiple, potentially hierarchically organized, CAs. The CAs' public keys are embedded in the emergency communication software. After this step, authorities can issue *identity certificates*. Furthermore, the overall PKI contains at least one authority that issues identity certificates to *unprivileged* users, i.e., citizens. In 4.61, the fire service maintains several hierarchically organized authorities. On the lowest hierarchical CA level, CAs issue identity certificates to staff members.

Identity certificates bind the public signing keys of users, which function as their unique network identifiers (see 4.9.5), to user properties. A vital user property is the *user role*, since it is important to assess the content of messages. For instance, citizens consider medical information more reliable if they originate from physicians rather than firefighters. Another essential property is the *authorization level* that indicates the permission level and trustworthiness of a user. 4.61 shows the identity certificate of a firefighter team leader, and depicts the authorization level of entities by their x-coordinate as well as stars in the certificate. In order to obtain an identity certificate, users must register with the CA and provide a proof of identity, e.g., using their identification card, phone number, or address. The identity proof is vital to hamper multi-registrations, where a single user obtains multiple identity certificates.

Since an adversary may obtain identity certificates, compromise user devices, or even infiltrate authorities, it is important that certificates can be revoked. SEDCOS implements certificate revocations via certificate revocation lists (CRLs) that are broadcasted with high priority in the network. It is distinguished between two different entities: authorities and users. An authority \mathcal{A} can revoke an entity \mathcal{E} if \mathcal{A} has a higher authorization level than \mathcal{E} , and there is a certificate chain (i.e., a chain of trust) between \mathcal{A} and \mathcal{E} . Upon the revocation of an authority, all certificates that the authority issued in the past and will issue in the future are regarded as invalid, withdrawing its power.

In case a user identity certificate is revoked, the respective user becomes an uncertified and unprivileged user, hence, loses its user role, authorization level, and any message transmission privileges (see 4.9.5).

4.9.5 Resilient Communication

In this subsection, first an overview of the communication protocol is given and then the design of the DoS-resistant buffer management is given.

Protocol Overview

A short overview of the used message protocol is given in the following.

Message Format and Types All SEDCOS messages have the same format and include the following fields: message *type*, sender and receiver *addresses*, *creation time* and *lifetime* (together yielding the time-to-live (TTL)), sender *signature*, and the optionally encrypted *payload*. It is emphasized that all header fields are *immutable*, that is, they are not changed in transit, thus, allowing the signature to protect the entire message. The message type can be:

- *Certificate Revocation Lists*;
- *Network control* with subtypes for acknowledgments and the device-to-device message exchange handshake;
- *Content* sent by users.

Acknowledgments are sent by the destination upon reception of a message.

Message Authenticity and Confidentiality Each user possesses a unique Elliptic Curve Digital Signature (ECDSA) *signature key pair*. The public part of the key serves as a unique addressable *network identifier*. Each outgoing message is signed using this key and can optionally be augmented with the identity certificate. Devices verify messages at each hop by checking the message signature and, if available, the sender's identity certificate; and discard them if any check fails. Hence, corrupted messages do not propagate in the network. To achieve data confidentiality, each user generates its own Elliptic Curve Integrated Encryption Scheme (ECIES) *encryption key pair* during initialization. Consequently, sending or receiving confidential messages requires the message payload to be encrypted with the public or decrypted with the private ECIES key of the receiving user.

Message Storage Each device reserves persistent memory for storing its own as well as others' messages. This memory space is referred to as the *buffer*. Its capacity C depends on the device capabilities and can be adjusted by the user. Efficiently managing the buffer is crucial for delivery reliability, as shown in 4.9.6.

Message Exchange When two devices discover each other via Bluetooth or Wi-Fi beacon frames, they connect to exchange messages. Currently *epidemic* dissemination is used, i.e., nodes try to exchange all carried messages. This introduces redundancy in the network, which helps when single nodes "disappear" (low battery or mobility).

Algorithm 1 Source-based Elastic Bucket Insertion

Require: $msg, buckets, C$

```

1:  $s \leftarrow$  source of  $msg$ 
2: if not  $buckets$  contains bucket for  $s$  then
3:   insert new empty bucket for  $msg$  in  $buckets$ ;
4: end if
5:  $B_s \leftarrow$  bucket from  $buckets$  for  $s$ ;
6: insert  $msg$  into  $B_s$ ;
7: while occupancy of  $buckets$  exceeds  $C$  do
8:    $\hat{B} \leftarrow$  bucket from  $buckets$  with the highest occupancy;
9:   remove message with the lowest rank from  $\hat{B}$ ;
10:  if  $\hat{B}$  is empty then
11:    remove  $\hat{B}$  from  $buckets$ ;
12:  end if
13: end while
    
```

However, due to limited buffer capacity and possibly short contact times (i. e., two cars passing each other), not all messages might be exchanged. Thus, messages are exchanged in the following order:

- messages destined for \mathcal{B} ,
- messages from *privileged* users (authorities),
- all other messages.

Source-based Elastic Buckets

Proper buffer management is essential to prevent resource starvation attacks such as flooding. Malicious nodes can easily exploit trivial implementations such as FIFO queues containing all messages to replace valid messages with bogus ones [156]. To counter such attacks, a novel buffer management strategy called *Source-based Elastic Buckets (SEB)* is employed that, by design, prevents valid messages from being purged during flooding attacks. The basic idea is that all messages from a source s are placed in an isolated bucket B such that messages from different sources cannot influence one another. SEB is fair in the sense that each bucket has a guaranteed *capacity* of $C_B = \lfloor C/n \rfloor$ where n is the number of currently allocated buckets (= number of source nodes currently carry messages from). The *occupancy* of a single source bucket O_B is subject to $O_B \in [0, C]$ and $\sum_s O_B \leq C$. If s does not exhaust its guaranteed capacity ($O_B < C_B$) because it has not sent “enough” messages, free capacity ($C_B - O_B$) is shared by other buckets requiring it. However, when s sends a message at a later point, overdrawn buckets ($O_B > C_B$) are emptied first. These *elastic* quotas allow full exploitation of local buffer capacities while maintaining strict message separation of different source nodes. Algorithm 1 shows SEB’s message insertion procedure: the underlying idea is that SEB inserts new messages in the appropriate (source) bucket and then drops messages from the highest occupant bucket until the total occupancy meets C . Note that a node will always try to make space for its messages by dropping its messages last. This is to ensure that there is at least one copy of every message in the network. However, if a device injects too many new messages (exceeding C), its buffer overflows, and SEB eventually has to drop own messages. Within each bucket, SEB prioritizes:

- security control messages (revocation certificates),
- network control messages (acknowledgments), and
- messages with the longest remaining TTL.

SEB's robustness relies on the fact that messages are source-authenticated and on the relatively high costs of acquiring new identities in the system. Without the latter costs, an attacker could assume multiple identities, flood the network with messages and, thus, hijack a disproportional amount of buffer capacity.

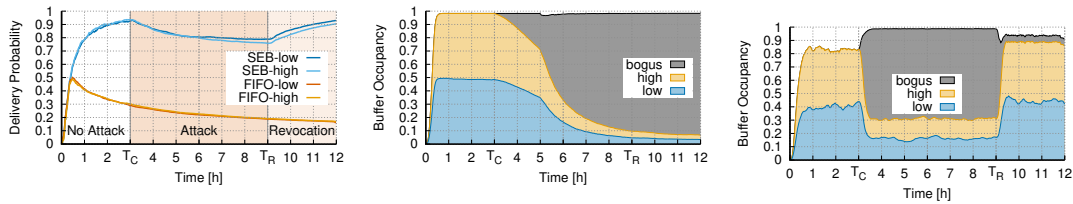
4.9.6 Experimental Evaluation

In this subsection, the impact of flooding attacks by several privileged devices (due to theft or compromise), and their eventual revocation from the system is evaluated.

Scenario

Three different user classes are considered with a total of 1000 nodes: 850 *citizens*, 100 *authorities*, and 50 *attackers*. Within each group, there are 5 % cars (10–50 km/h), all others move at walking speed (1.8–4.5 km/h). Citizens can transmit *low*-priority messages, while authorities sent with *high* priority (interval: 15–25 s). SEB is compared to a classic FIFO queue, both using epidemic routing. The buffer capacity C is 5 MB. The ONE simulator v1.6.0 [157] as well as the default Helsinki map for the experiments is used and the results are averaged over ten differently-seeded runs. Bluetooth communication with 2 Mbit/s and a range of 10 m is assumed.

Flooding Attack and Revocation



(a) Delivery rate of valid messages over time. (b) Buffer occupancy with FIFO (c) Buffer occupancy with SEB

Figure 4.62: Flooding attack and revocation.

The attack starts at $T_C = 3$ h and the revocation certificate is issued at $T_R = 9$ h.

The impact of an attacker being able to compromise privileged devices is evaluated. In this case, the attacker is able to inject *bogus* high-priority messages into the network, thus, increasing their chances to remain in the nodes' buffers for a long time. In this experiment, two events occur: at $T_C = 3$ h, the attackers start the flooding attack using compromised devices; and after a reaction time of 6 h at $T_R = 9$ h, an authority issues and injects the revocation certificate into the network. When a node receives the revocation certificate, it drops all messages it carries from the revoked nodes and blacklists future messages by those nodes. Attackers ignore revocation certificates.

4 Disruption-tolerant Device-to-Device Emergency Communication

Before the attack Figure 4.62a shows the delivered benign messages (*low* and *high*) over time. In the beginning, *SEB* quickly starts to successfully deliver most messages. *FIFO* follows the same start-up behavior but is not able to keep up from the 30 minutes mark. After 30 minutes, buffers are filled up (Fig. 4.62b) and the lack of proper buffer management leads to poor delivery performance.

During the attack At the start of the attack, *SEB*'s delivery probability remains almost unaffected by the flooding attack even though buffers quickly fill up to 70 % with *bogus* messages (Fig. 4.62c). The decrease in delivery probability is only about 10 % (Fig. 4.62a) demonstrating the effectiveness of the source-based elastic buckets: they assure that *bogus* messages cannot overtake the entire buffer capacity. *FIFO* reacts less visibly to the attack since the delivery probability is already low at T_C (4.62a). Yet, the impact is apparent in 4.62b where bogus messages steadily take up more buffer capacity, leading to continuously decreasing delivery probability.

Aftermath *FIFO* does not recover from the attack after T_R , since only a few nodes receive the revocation certificate due to the lack of a prioritization mechanism. With *SEB*, the revocation certificate propagates quickly throughout the network: half of the nodes are informed within 12 minutes while full network penetration is reached in less than one hour. The drop of bogus message buffer occupancy shortly after T_R indicates the revocation certificate's effect (4.62c). As the certificate propagates in the network, buffer occupancy restores to a state similar to $t < T_C$. Nevertheless, a small fraction of bogus messages remains in the network since attackers ignore the revocation certificate and keep their messages in their buffers.

4.9.7 Conclusion

A secure and reliable communication system is essential for effective disaster response. *SEDCOS*, a system that enables secure and reliable disruption-tolerant emergency communication on commodity mobile devices, was presented. It is the first secure emergency communication system that enables the exclusion of adversaries while providing authentic and confidential group communication. Under DoS attacks, *SEDCOS* increases the message delivery rate by a factor of 6 compared to a contemporary DTN protocol. Finally, *SEDCOS* provides a timely revocation (less than one hour) for withdrawing any power of an insider adversary.

While the solutions described in this section proved to be effective, they have only been evaluated in a simulator. In the future, integration into a real DTN middleware, as used in the previous sections, would be a valuable task. Furthermore, different ways of attestation, local and remote, for handing out identity certificates during a disaster should be investigated.

4.10 Summary

In Section 4.2, *MiniWorld* was presented, a novel distributed network emulator. It is based on full virtualization using QEMU/KVM, offers three network backends for emulating both wired and wireless communication, and provides several mobility patterns as well as distance-based link quality models. A snapshot boot mode is offered for accelerated booting of identical environments and repeating emulation runs. To decrease runtimes, *MiniWorld* supports distributed emulation across multiple computers, based on a resource-aware virtual machine (VM) scheduler. Experimental results demonstrate the performance of *MiniWorld* with respect to VM boot times, network bandwidth, round trip times, and topology switching times.

An in-depth experimental evaluation of Serval for various network setups and usage patterns, including simulated long term use was performed in Section 4.3. The focus of the evaluation was on the delay-tolerant aspects of Serval, providing insights into the scenarios where Serval can be deployed with satisfactory quality and performance characteristics. Furthermore, since mobile phones have a limited battery capacity, a closer look is taken at the battery drain resulting from using Serval over different communication links, such as WiFi and Bluetooth. Despite minor shortcomings, it was shown that Serval is a flexible foundation for data and message dissemination in various environments.

Several approaches to realize dynamic announcement intervals that facilitate fast reception from at least one other node while trying to keep the overall communication overhead as low as possible were developed in Section 4.4. Experimental results in terms of performance properties and energy consumption were presented to illustrate the benefits of dynamic announcement intervals in wireless on-demand networks.

In Section 4.5, *DTN-RPC*, a new approach to provide RPCs for DTN environments, was presented. *DTN-RPC* relies on (a) control and data channels to cope with potentially short contact durations in DTN where large amounts of data cannot be transmitted, (b) explicit and implicit modes to address remote servers, (c) Non-DTN and DTN transport protocols for issuing calls and receiving results, and (d) predicates that servers check to decide whether a procedure should be executed. The implementation of *DTN-RPC* is based on Serval. The experimental results indicate that the measured CPU and network overheads for *DTN-RPC* are reasonably low so that it can be executed on smartphones or routers, and that the round-trip times and the number of successful RPCs are highly satisfactory in dynamic networks with unstable links.

In Section 4.6 a novel, low-cost yet flexible and powerful hardware/software platform for sensing, computation, and infrastructureless wireless communication was presented. Since energy consumption is a key issue for autonomous operation, the power requirements of various computing platforms, sensors and radio link technologies were investigated. Furthermore, several license-free radios (WiFi, Bluetooth, LoRa 433 MHz and 868 Mhz) were experimentally evaluated regarding their real-world communication ranges. To avoid wasting precious communication resources, machine learning and image-based concept detection was applied on-device to remove irrelevant data prior to sending. Neural compute sticks were also evaluated to improve performance and optimize their energy consumption in this specific scenario. The proposed hardware/software platform consists of small, low-power sensor nodes based on microcontrollers, larger single board computer relay nodes that can also preprocess data, and Bluetooth

4 Disruption-tolerant Device-to-Device Emergency Communication

Low Energy enabled smartphone add-ons to give mobile devices access to long range communication technology. Finally, a Linux distribution tailored to the specific needs in this application area is presented that makes deploying new relay nodes easy even for non-specialists.

SmartFace, a novel approach to perform face detection locally on mobile devices in an efficient manner, was presented in Section 4.7. The approach relies on a two-stage combination of existing face detection algorithms, enhanced by region of interest selection, color space/depth reduction, resolution scaling, face size definition, image scaling, image cropping, and bounding box scaling. Experimental results indicate that the proposed approach improves both the overall face detection rate and the overall runtime compared to the individual face detection algorithms used alone. It also reduces the amount of data that needs to be stored on disk and sent over the network.

UV4EV was presented in Section 4.8, a novel approach to emergency communication where semi-autonomous UGVs and UAVs cooperate with humans to dynamically form communication islands and establish communication bridges between these islands. Humans typically form an island with their mobile devices if they are in physical proximity; UGVs and UAVs extend an island's range by carrying data to a neighboring island. The proposed approach uses delay/disruption-tolerant networking for non-time critical tasks and direct mesh connections for prioritized tasks that require real-time feedback. The developed communication platform runs on rescue robots, commodity mobile devices, and various drones, and supports our operations and control center software for disaster management.

SEDCOS, a secure device-to-device communication system for disaster scenarios, was presented in Section 4.9. It mitigates flooding DoS attacks and offers role revocation for detected adversaries to withdraw their permissions. The effectiveness of *SEDCOS* was demonstrated by large-scale network simulations.

The research in this chapter shows that even in uncertain and challenged network conditions, such as the ones found during emergency scenarios, common applications and services can be delivered through new ways. By relying on D2D communication together with DTN, many of the challenges can be overcome.

5 Security Vulnerability Analysis of Mobile Apps

5.1 Introduction

While the previous chapter focused on disruption-tolerant device-to-device emergency communication in general, this chapter provides an in-depth insight into the security aspects of existing emergency apps and their communication. Most apps found on smartphones use network resources, often through HTTP(S), or store valuable data, such as images, contacts, bank credentials, cryptographic identities and messages. Therefore, analyzing emergency apps for possible vulnerabilities is critical for secure emergency communication.

One fundamental technology to secure data-in-motion is the use of SSL/TLS. In Section 5.2, a study of the use and implementation security state of SSL in Android apps is presented.

To ease the process of such audits, a flexible framework for distributed static analysis, called *AndroLyze*, is presented in Section 5.3. Furthermore, the use of cryptographic functions in about 40,000 APKs is investigated.

In Section 5.4 *Dynalize*, a Platform-as-a-Service cloud for dynamic analysis of Android apps, is proposed. It completes the flexible toolkit, together with *AndroLyze*, for large-scale analysis of mobile apps.

The experience from the previous sections is used to give a security report on the most commonly used emergency communication apps currently available for Android. The results of this audit are presented in Section 5.5.

*AndroLyze*¹ as well as *Dynalize*² have been published on github.

¹<https://github.com/nachtmaar/androlyze>

²<https://github.com/umr-ds/dynalize>

5.2 TLS Usage in Android Apps

5.2.1 Introduction

Currently, Android is the most used smartphone operating system in the world, with a market share of 48%³ and over 400,000 applications (apps) available in the Google Play Market⁴, almost doubling the number of apps in only six months.⁵ Android apps have been installed over 10 billion times⁶ and cover a vast range of categories from games and entertainment to financial and business services. Unlike the “walled garden approach” of Apple’s App Store, Android software development and the Google Play Market are relatively open and unrestricted. This offers both developers and users more flexibility and freedom, but also creates significant security challenges.

The coarse permission system [158] and over-privileging of applications [159] can lead to exploitable applications. Consequently, several efforts have been made to investigate privilege problems in Android apps [158], [160]–[163]. Enck et al. introduced TaintDroid [164] to track privacy-related information flows to discover such (semi-)malicious apps. Bugiel et al. [162] showed that colluding malicious apps can facilitate information leakage. Furthermore, Enck et al. analyzed 1,100 Android apps for malicious activity and detected widespread use of privacy-related information such as IMEI, IMSI, and ICC-ID for “cookie-esque” tracking. However, no other malicious activities were found, in particular no exploitable vulnerabilities that could have lead to malicious control of a smartphone were observed [165].

In this section, instead of focusing on malicious apps, the potential security threats posed by benign Android apps that legitimately process privacy-related user data, such as log-in credentials, personal documents, contacts, financial data, messages, pictures or videos is investigated. Many of these apps communicate over the Internet for legitimate reasons and thus request and require the INTERNET permission. It is then necessary to trust that the app adequately protects sensitive data when transmitting it via the Internet.

The most common approach to protect data during communication on the Android platform is to use the Secure Sockets Layer (SSL) or Transport Layer Security (TLS) protocols.⁷ To evaluate the state of SSL use in Android apps, 13,500 popular free apps were downloaded from Google’s Play Market and their properties were studied with respect to the usage of SSL. In particular, the apps’ vulnerability against Man-in-the-Middle (MITM) attacks due to the inadequate or incorrect use of SSL was studied.

For this purpose, MalloDroid was created, an Androguard⁸ extension that performs static code analysis to a) analyze the networking API calls and extract valid HTTP(S) URLs from the decompiled apps; b) check the validity of the SSL certificates of all extracted HTTPS hosts; and c) identify apps that contain API calls that differ from Android’s default SSL usage, e. g., contain non-default trust managers, SSL socket factories or hostname verifiers with permissive verification strategies. Based on the results of the

³<https://bit.ly/L4c8Ky>

⁴<https://bit.ly/xr7WET>

⁵<https://bit.ly/rhJxf2>

⁶<https://bit.ly/H1qGta>

⁷Android supports both SSL and TLS; for brevity, it will be referred to both protocols as SSL. The issues described in this work affect both SSL and TLS in the same way.

⁸<http://code.google.com/p/androguard/>

static code analysis, 100 apps were selected for manual audits to investigate various forms of SSL use and misuse: accepting all SSL certificates, allowing all hostnames regardless of the certificate's Common Name (CN), neglecting precautions against SSL stripping, trusting all available Certificate Authorities (CAs), not using SSL pinning, and misinforming users about SSL usage.

Furthermore, the visibility and awareness of SSL security in the context of Android apps was studied. In Android, the user of an app has no guarantee that an app uses SSL and also gets no feedback from the Android operating system whether SSL is used during communication or not. It is entirely up to the app to use SSL and to (mis)inform the user about the security of the connection. However, even when apps present warnings and security indicators, users need to see and interpret them correctly. The users' perceptions concerning these warnings and indicators were investigated in an online survey. Finally, several countermeasures that could help to alleviate the problems discovered in the course of this work are discussed.

The results of the investigations can be summarized as follows:

- 1,074 apps contain SSL specific code that either accepts all certificates or all hostnames for a certificate and thus are potentially vulnerable to MITM attacks.
- 41 of the 100 apps selected for manual audit were vulnerable to MITM attacks due to various forms of SSL misuse.
- The cumulative install base of the apps with confirmed vulnerabilities against MITM attacks lies between 39.5 and 185 million users, according to Google's Play Market.⁹ This number includes 3 apps with install bases between 10 and 50 million users each.
- From these 41 apps, it was possible to – for example – capture credentials for American Express, Diners Club, Paypal, bank accounts, Facebook, Twitter, Google, Yahoo, Microsoft Live ID, Box, WordPress, remote control servers, arbitrary email accounts, and IBM Sametime.
- Virus signatures were injected into an anti-virus app to detect arbitrary apps as a virus or disable virus detection completely.
- It was possible to remotely inject and execute code in an app created by a vulnerable app building framework.
- 378 (50.1%) of the 754 Android users participating in the online survey did not judge the security state of a browser session correctly.
- 419 (55.6%) of the 754 participants had not seen a certificate warning before and typically rated the risk they were warned against as medium to low.

Parts of this section have been published in [9].

5.2.2 Background

The focus of the investigation is the inadequate use of SSL in Android apps. In this subsection, a brief overview of how SSL is used in Android is given and how MITM attacks can be launched against broken SSL connections in the context of this research.

⁹Google's Play Market does not give a precise number of installs, instead giving a range. The actual number is likely to be larger, since alternative app markets for Android also contribute to the install base.

SSL

The Secure Sockets Layer (SSL) and its successor, Transport Layer Security (TLS), are cryptographic protocols that were introduced to protect network communication from eavesdropping and tampering. To establish a secure connection, a client must securely gain access to the public key of the server. In most client/server setups, the server obtains an X.509 certificate that contains the server's public key and is signed by a Certificate Authority (CA). When the client connects to the server, the certificate is transferred to the client. The client must then validate the certificate.¹⁰ However, validation checks are not a central part of the SSL and X.509 standards. Recommendations are given, but the actual implementation is left to the application developer.

The basic validation checks include: a) does the subject (CN) of the certificate match the destination selected by the client?; b) is the signing CA a trusted CA?; c) is the signature correct?; and d) is the certificate valid in terms of its time of expiry? Additionally, revocation of a certificate and its corresponding certificate chain should be checked, but downloading Certificate Revocation Lists (CRLs) or using the Online Certificate Status Protocol (OCSP)¹¹ is often omitted. The open nature of the standard specification has several pitfalls, both on a technical and a human level. Therefore, the evaluations in the remainder of this work are based on examining the four validation checks listed above.

Android & SSL

The Android 4.0 SDK offers several convenient ways to access the network. The `java.net`, `javax.net`, `android.net` and `org.apache.http` packages can be used to create (server) sockets or HTTP(S) connections. The `org.webkit` package provides access to web browser functionality. In general, Android allows apps to customize SSL usage – i. e., developers must ensure that they use SSL correctly for the intended usage and threat environment. Hence, the following (mis-) use cases can arise and can cause an app to transmit sensitive information over a potentially broken SSL channel:

Trusting all Certificates. The `TrustManager` interface can be implemented to trust all certificates, irrespective of who signed them or even for what subject they were issued.

Allowing all Hostnames. It is possible to forgo checks of whether the certificate was issued for this address or not, i. e., when accessing the server `example.com`, a certificate issued for `some-other-domain.com` is accepted.

Trusting many CAs. This is not necessarily a flaw, but Android 4.0 trusts 134 CA root certificates per default. Due to the attacks on several CAs in 2011, the problem of the large number of trusted CAs is actively debated.¹²

Mixed Mode/No SSL. App developers are free to mix secure and insecure connections in the same app or not use SSL at all. This is not directly a SSL issue, but it is relevant to mention that there are no outward signs and no possibility for a common app user to check whether a secure connection is being used. This opens the door for attacks such as SSL Stripping [166], [167] or tools like Firesheep.¹³

¹⁰<https://tools.ietf.org/html/rfc5280>

¹¹<https://tools.ietf.org/html/rfc2560>

¹²<https://bit.ly/g0dH34>

¹³<https://codebutler.com/firesheep>

On the other hand, Android's flexibility in terms of SSL handling allows advanced features to be implemented. One important example is SSL Pinning¹⁴, in which either a (smaller) custom list of trusted CAs or even a custom list of specific certificates is used. Android does not offer SSL pinning capabilities out of the box. However, it is possible to create a custom trust manager to implement SSL pinning.¹⁵

The use of an SSL channel, even under the conditions described above, is still more secure than using only plain HTTP against a passive attacker. An active MITM attack is required for an attacker to subvert an SSL channel and is described below.

MITM Attack

In a MITM attack (MITMA), the attacker is in a position to intercept messages sent between communication partners. In a passive MITMA, the attacker can only eavesdrop on the communication (attacker label: Eve), and in an active MITMA, the attacker can also tamper with the communication (attacker label: Mallory). MITMAs against mobile devices are somewhat easier to execute than against traditional desktop computers, since the use of mobile devices frequently occurs in changing and untrusted environments. Specifically, the use of open access points [32] and the evil twin attack [33] make MITMAs against mobile devices a serious threat.

SSL is fundamentally capable of preventing both Eve and Mallory from executing their attacks. However, the cases described above open up attack vectors for both Eve and Mallory. Trivially, the mixed mode/no SSL case allows Eve to eavesdrop on non-protected communication.

SSL Stripping is another method by which a MITMA can be launched against an SSL connection, exploiting apps that use a mix of HTTP and HTTPS. SSL Stripping relies on the fact that many SSL connections are established by clicking on a link in or being redirected from a non-SSL-protected site. During SSL Stripping, Mallory replaces *https://* links in the non protected sites with insecure *http://* links. Thus, unless the user notices that the links have been tampered with Mallory can circumvent SSL protection altogether. This attack is mainly relevant to browser apps or apps using Android's WebView.

5.2.3 Related Work

So far, there is no in-depth study of SSL usage and security on Android phones to date. Thus, the discussion of related work is divided into two parts: related work concerning Android security and a selection of SSL security work relevant for this research.

Android Security

There have been several efforts to investigate Android permissions and unwanted or malicious information flows, such as the work presented by Enck et al. [158], [164], Porter Felt et al. [159], [160], Davi et al. [161], Bugiel et al. [162], Nauman et al. [168] and Egner et al. [169]. These works have in common that they study how permissions can be abused and how this abuse can be prevented. Their scope does not include the study of SSL issues, and the proposed countermeasures do not mitigate the threats

¹⁴<https://bit.ly/qugGtH>

¹⁵<https://bit.ly/v55mxn>

presented here. The vulnerabilities studied in this thesis are based on weaknesses in the design and use of SSL and HTTPS in Android apps. Since the permissions used by the apps during SSL connection establishment are legitimate and necessary, the current permissions-based countermeasures would not help.

There are several good overviews of the Android security model and threat landscape, such as Vidas et al. [170], Shabatai [171] et al. and Enck et al. [165], [172]. These papers do not discuss the vulnerability of SSL or HTTPS on Android. Enck et al. [165] does mention that some apps use sockets directly, bearing the potential for vulnerabilities, but no malicious use was found (cf. [165], Finding 13). The investigation in this research shows that there are several SSL-related vulnerabilities in Android apps, endangering millions of users.

McDaniel et al. [173] and Zhou et al. [174] also mainly focus on malicious apps in their work on the security issues associated with the app market model of software deployment. The heuristics of DroidRanger [174] could be extended to detect the vulnerabilities uncovered in this work.

SSL Security

A good overview of current SSL problems can be found in Moxi Marlinspike's Black Hat talks [166], [167]. The talks cover issues of security indicators, Common Name (CN) mismatches and the large number of trusted CAs and intermediate CAs. Marlinspike also introduces the SSL Stripping attack. The fact that many HTTPS connections are initiated by clicking a link or via redirects is particularly relevant for mobile devices, since the MITMA needed for SSL Stripping is easier to execute against mobile devices [32], [33] and the visual indicators are hard to see on mobile devices.

Shin et al. [175] study the problem of SSL Stripping for desktop browsers and present a visual-security-cue-based approach to hinder SSL Stripping in this environment. They also highlight the particular problem of this type of attack in the mobile environment and suggest that it should be studied in more detail.

Egelman et al. [176] and Sunshine et al. [177] both study the effectiveness of browser warnings, showing that their effectiveness is limited and that there are significant usability issues. Although both of these studies were conducted in a desktop environment, the same caveats need to be considered for mobile devices. In this research, a first online survey was conducted to gauge the awareness and effectiveness of browser certificate warnings and HTTPS visual security indicators on Android.

5.2.4 Evaluating Android SSL Usage

The study of Android SSL security encompasses 13,500 popular free apps from Google's Play Market. MalloDroid, an extension of the Androguard reverse engineering framework, was build to automatically perform the following steps of static code analysis:

Permissions. MalloDroid checks which apps *request the INTERNET permission*, which apps *actually contain INTERNET permission-related API calls* and which apps *additionally request and use privacy-related permissions* (c.f. [159]).

Networking API Calls. MalloDroid analyzes the use of *HTTP transport* and *Non-HTTP transport* (e. g., direct socket connections).

HTTP vs. HTTPS. MalloDroid checks the validity of URLs found in apps and groups the apps into *HTTP only*, *mixed mode (HTTP and HTTPS)* and *HTTPS only*.

HTTPS Available. MalloDroid tries to establish a secure connection to HTTP URLs found in apps.

Deployed Certificates. MalloDroid downloads and evaluates SSL certificates of hosts referenced in apps.

SSL Validation. MalloDroid examines apps with respect to inadequate SSL validation (e.g., apps containing code that allows all hostnames or accepts all certificates).

12,534 (92.84%) of the apps in the test set request the network permission `android.permission.INTERNET`. 11,938 (88.42%) apps actually perform networking related API calls. 6,907 (51.16%) of the apps in the sample use the `INTERNET` permission in addition to permissions to access privacy related information such as the users' calendars, contacts, browser histories, profile information, social streams, short messages or exact geographic locations. This subset of apps has the potential to transfer privacy-related information via the Internet. This subset does not include apps such as banking, business, email, social networking or instant messaging apps that intrinsically contain privacy-relevant information without requiring additional permissions.

It was found that 91.7% of all networking API calls are related to HTTP(S). Therefore, the decision was made to focus the further analysis on the usage of HTTP(S). To find out whether an app communicates via HTTP, HTTPS, or both, MalloDroid analyzes HTTP(S) specific API calls and extracts URLs from the decompiled apps.

HTTP vs. HTTPS

MalloDroid extracted 254,022 URLs. It can be configured to remove certain types of URLs for specific analysis. For this study, the 58,617 URLs pointing to namespace descriptors and images were removed, since these typically are not used to transmit sensitive user information. The remaining 195,405 URLs pointed to 25,975 unique hosts. 29,685 of the URLs (15.2%) pointing to 1,725 unique hosts (6.6%) are HTTPS URLs. Then it was analyzed how many of the hosts referenced in HTTP URLs could also have been accessed using HTTPS.

76,435 URLs (39.1%) pointing to 4,526 hosts (17.4%) allowed a valid HTTPS connection to be established, using Android's default trust roots and validation behavior of current browsers. This means that 9,934 (73.6%) of all 13,500 tested apps could have used HTTPS instead of HTTP with minimal effort by adding a single character to the target URLs. It was found that 6,214 (46.0%) of the apps contain HTTPS and HTTP URLs simultaneously and 5,810 (43.0%) do not contain HTTPS URLs at all. Only 111 apps (0.8%) exclusively contained HTTPS URLs.

For a more detailed investigation, it was looked at the top 50 hosts, ranked by number of occurrences. This group mainly consists of advertising companies and social networking sites. These two categories account for 37.9% of the total URLs found, and the hosts are contained in 9,815 (78.3%) of the apps that request the `INTERNET` permission.

Table 5.1 presents an overview of the top 10 hosts. The URLs pointing to these hosts suggest they are often used for Web Service API calls, authentication and fetching/sending user or app information. Especially in the case of ad networks that

5 Security Vulnerability Analysis of Mobile Apps

collect phone identifiers and geolocations [164] and social networks that transport user-generated content, the contained information is potentially sensitive.

Table 5.1: The top 10 hosts used in all extracted URLs and their SSL availability, total number of URLs and number of HTTPS URLs pointing to that host.

Host	has SSL	# URLs	# HTTPS
market.android.com	✓	6,254	3,217
api.airpush.com	✓	5,551	0
a.admob.com	✓	4,299	0
ws.tapjoyads.com	✓	3,410	3399
api.twitter.com	✓	3,220	768
data.flurry.com	✓	3,156	1,578
data.mobclix.com	✓	2,975	0
ad.flurry.com	✓	2,550	0
twitter.com	✓	2,410	129
graph.facebook.com	✓	2,141	1,941

34 of the top 50 hosts offer all their API calls via HTTPS, but none is accessed exclusively via HTTPS. Of all the URLs pointing to the top 50 hosts, 22.1% used HTTPS, 61.0% could have used HTTPS by substituting *http://* with *https://*, and 16.9% had to use HTTP because HTTPS was not available. The hosts [facebook.com](https://www.facebook.com) and [tapjoyads.com](https://www.tapjoyads.com) are positive examples, since the majority of the URLs found for these two hosts already use HTTPS.

Deployed SSL Certificates

To analyze the validity of the certificates used by HTTPS hosts, the SSL certificates for all HTTPS hosts extracted from our app test set were downloaded, yielding 1,887 unique SSL certificates. Of these certificates, 162 (8.59%) failed the verification of Android's default SSL certificate verification strategies, i. e., 668 apps contain HTTPS URLs pointing to hosts with certificates that could not be validated with the default strategies. 42 (2.22%) of these certificates failed SSL verification because they were self-signed, i. e., HTTPS links to self-signed certificates are included in 271 apps. 21 (1.11%) of these certificates were already expired, i. e., 43 apps contain HTTPS links to hosts with expired SSL certificates.

For hostname verification, two different strategies were applied that are also available in Android: the *BrowserCompatHostnameVerifier*¹⁶ and the *StrictHostnameVerifier*¹⁷ strategy. 112 (5.94%) certificates were found that did not pass strict hostname verification, of which 100 certificates also did not pass the browser compatible hostname verification. Mapping these certificates to apps revealed that 332 apps contained HTTPS URLs with hostnames failing the *BrowserCompatHostnameVerifier* strategy.

Overall, 142 authorities signed 1,887 certificates. For 45 (2.38%) certificates, no valid certification paths could be found, i. e., these certificates were signed by authorities not reachable via the default trust anchors. These certificates are used by 46 apps. All in all,

¹⁶<https://bit.ly/IKR9cD>

¹⁷<https://bit.ly/Ixy9kr>

394 apps include HTTPS URLs for hosts that have certificates that are either expired, self-signed, have mismatching CNs or are signed by non-default-trusted CAs.

Custom SSL Validation

Using MalloDroid, 1,074 apps (17.28% of all apps that contain HTTPS URLs) were found that include code that either bypasses effective SSL verification completely by accepting all certificates (790 apps) or that contain code that accepts all hostnames for a certificate as long as a trusted CA signed the certificate (284 apps).

While an app developer wishing to accept all SSL certificates must implement the TrustManager interface and/or extend the SSLSocketFactory class, allowing all hostnames only requires the use of the AllowAllHostnameVerifier class from the `org.apache.http.conn.ssl` package that is included in 453 apps. Additionally, MalloDroid found a FakeHostnameVerifier, NaiveHostnameVerifier and AcceptAllHostnameVerifier class that can be used in the same way.

To understand how apps use ‘customized’ SSL implementations, it was searched for apps that contain non-default trust managers, SSL socket factories and hostname verifiers differing from the BrowserCompatHostnameVerifier strategy. Here 86 custom trust managers and SSL socket factories were found in 878 apps. More critically, the analysis also discovered 22 classes implementing the TrustManager interface and 16 classes extending the SSLSocketFactory that accept all SSL certificates. Table 5.2 shows which broken trust managers and SSL socket factories were found.

Table 5.2: Trust Managers & Socket Factories that trust all certificates (suffixes omitted to fit the page)

Trust Managers	SSL Socket Factories
AcceptAllTrustM	AcceptAllSSLSocketF
AllTrustM	AllTrustingSSLSocketF
DummyTrustM	AllTrustSSLSocketF
EasyX509TrustM	AllSSLSocketF
FakeTrustM	DummySSLSocketF
FakeX509TrustM	EasySSLSocketF
FullX509TrustM	FakeSSLSocketF
NaiveTrustM	InsecureSSLSocketF
NonValidatingTrustM	NonValidatingSSLSocketF
NullTrustM	NaiveSslSocketF
OpenTrustM	SimpleSSLSocketF
PermissiveX509TrustM	SSLSocketFUntrustedCert
SimpleTrustM	SSLUntrustedSocketF
SimpleX509TrustM	TrustAllSSLSocketF
TrivialTrustM	TrustEveryoneSocketF
TrustAllManager	NaiveTrustManagerF
TrustAllTrustM	LazySSLSocketF
TrustAnyCertTrustM	UnsecureTrustManagerF
UnsafeX509TrustM	
VoidTrustM	

This small number of critical classes affects a large number of apps. Many of the found classes belong to libraries and frameworks that are used by many apps. 313 apps contained calls to the `NaiveTrustManager` class that is provided by a crash report library.¹⁸ In 90 apps, MalloDroid found the `NonValidatingTrustManager` class provided by an SDK¹⁹ for developing mobile apps for different platforms with just a single codebase. The `PermissiveX509TrustManager`²⁰, found in a library for sending different kinds of push notifications to Android devices, is included in 76 apps. Finally, in 78 apps, MalloDroid found a `SSLSocketFactory` provided by a developer library²¹ that accepts all certificates. The library is intended to support developers to write well designed software and promotes itself as a library for super-easy and robust networking. Using any of the above Trust Managers or Socket Factories results in the app trusting all certificates.

5.2.5 MITMA Study

The static code analysis presented above only shows the potential for security problems. The fact that code for insecure SSL is present in an app does not necessarily mean that it is used or that sensitive information is passed along it. Even more detailed automated code analysis, such as *control flow analysis*, *data flow analysis*, *structural analysis* and *semantic analysis* cannot guarantee that all uses are correctly identified [165]. Thus, the decision was made to conduct a more detailed manual study to find out what sort of information is actually sent via these potentially broken SSL communication channels, by installing apps on a real phone and executing an active MITMA against the apps. For this part of the study, the search was narrowed down to apps from the Finance, Business, Communication, Social and Tools categories, where one can suspect a higher amount of privacy-related information and a higher motivation to protect the information. In this test set, there are 266 apps containing broken SSL or hostname verifiers (Finance: 45, Social: 94, Communication: 49, Business: 60, Tools: 18). These apps were ranked based on their number of downloads and selected the top 100 apps for manual auditing. Additionally, 10 high profile apps (large install base, popular services) were cherry-picked that contained no SSL-related API calls but contained potentially sensitive information, to see whether this information was actually sent in the clear or if some protection mechanism other than SSL was involved.

Test Environment

For the manual app auditing process, a Samsung Galaxy Nexus smartphone with Android 4.0 Ice Cream Sandwich was used. The potentially vulnerable apps were installed on the phone and a WiFi access point with a MITM SSL proxy was set up. Depending on the vulnerability to be examined, the SSL proxy was equipped either with a self-signed certificate or with one that was signed by a trusted CA, but for an unrelated hostname.

Of the 100 apps selected for manual audit, 41 apps proved to have exploitable vulnerabilities. It was possible to gather bank account information, payment credentials

¹⁸Application Crash Report for Android (ACRA) library [org.acra.util.NaiveTrustManager](#)

¹⁹Titanium Mobile [ti.modules.titanium.network.NonValidatingTrustManager](#)

²⁰Urban Airship library [client.ssl.PermissiveX509TrustManager](#)

²¹Droid-Fu library [com.github.droidfu.http.ssl.EasySSLSocketFactory](#)

for PayPal, American Express and others. Furthermore, Facebook, email and cloud storage credentials and messages were leaked, access to IP cameras was gained and control channels for apps and remote servers could be subverted. According to Google's Play Market, the combined install base of the vulnerable apps in our test set of 100 apps was between 39.5 and 185 million users at the time of writing. In the following, the findings are briefly discussed to illustrate the scope of the problem.

Trusting All Certificates

21 apps among the 100 selected apps fell into this category. The MITMA proxy was given a self-signed certificate for the attack. The apps leaked information such as login credentials, webcam access or banking data. One noteworthy contender was a generic online banking app.²² The app uses separate classes for each bank containing different trust manager implementations. 24 of the 43 banks supported were not protected from the MITMA. The app also leaks login credentials for American Express, Diners Club and Paypal. The Google Play Market reports an install base between 100,000 and half a million users. A further app in this category offers instant messaging for the Windows Live Messenger service²³. The app has an install base of 10 to 50 million users and is listed in the top 20 apps for the communication category in the Google Play Market (as of April 30th, 2012). Username and password are both sent via a broken SSL channel and were sniffed during the attack. This effectively gives an attacker full access to a Windows Live account that can be used for email, messaging or Microsoft's SkyDrive cloud storage. Also in this category, a browser²⁴ with an install base between 500,000 and one million users was found. The browser does not correctly handle SSL at all, i.e., it accepts an arbitrary certificate for every website the user visits and hence leaks whatever data the user enters. All three apps do not provide any SSL control or configuration options for the user. None of the other apps in this category showed warning messages to the user while the MITMA was being executed.

Allowing All Hostnames

The second category of apps analyzed is the group of 20 apps that accepted certificates irrespective of the subject name, i.e., if the app wants to connect to <https://www.paypal.com>, it would also accept a certificate issued to some-domain.com. A certificate for an unrelated domain signed by startSSL²⁵ was used for the attacks in this category. The apps leaked information such as credentials for different services, emails, text messages, contact data, bitcoin miner api keys, premium content or access to online meetings. A particularly interesting find was an anti-virus app that updated its virus signatures file via a broken SSL connection. Since it seems that the connection was considered secure, no further validation of the signature files is performed by the app. Thus it was possible to feed a modified signature file to the anti-virus engine. First, an empty signature database was sent which was accepted, effectively turning off the anti-virus protection without informing the user. In a second attack, a virus signature for the anti-virus app itself was created and then sent it to the phone. This signature

²²com.liato.bankdroid

²³[miyowa.android.microsoft.wlm](https://com.miyowa.android.microsoft.wlm)

²⁴[sui.m](https://com.sui.m)

²⁵<https://www.startssl.com/>

5 Security Vulnerability Analysis of Mobile Apps

was accepted by the app, which then recognized itself as a virus and recommended to delete itself, which it also did. Figure 5.1 shows a screenshot of the result of this attack. This is a very stark reminder that defense in depth is an important security principle. Since the SSL connection was deemed secure, no further checks were performed to determine whether the signature files were legitimate. The app has an install base of 500,000 to one million users.²⁶

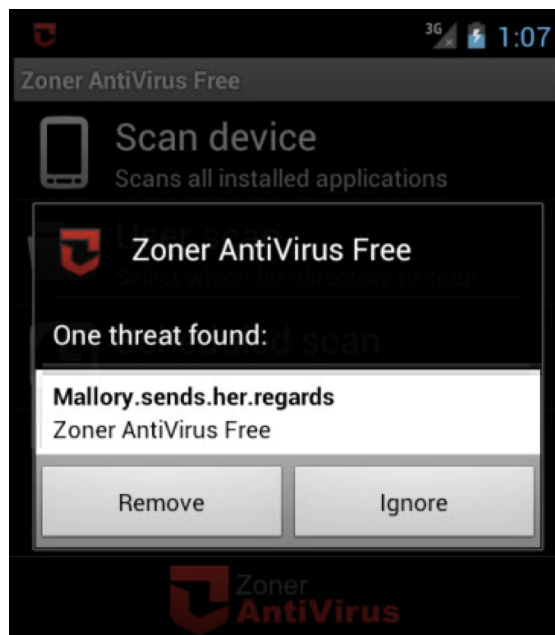


Figure 5.1: After injecting a virus signature database via a MITM attack over broken SSL, the AntiVirus app recognized itself as a virus and recommended to delete the detected malware.

A second example in this category is an app that offers "Simple and Secure" cloud-based data sharing.²⁷ According to the website, the app is used by 82% of the FORTUNE 500 companies to share documents. It has an install base between 1 and 5 million users. While the app offers simple sharing, it leaks the login credentials during the MITMA. One interesting finding in this app was that the login credentials were leaked from a broken SSL channel while up- and downloads of files were properly secured. However, using the login credentials obtained from the broken channel is sufficient to hijack an account and access the data anyway.

A third example is a client app for a popular Web 2.0 site²⁸ with an install base of 500,000 to 1 million users. When using a Facebook or Google account for login, the app initiates OAuth login sequences and leaks Facebook or Google login credentials.

Also a very popular cross-platform messaging service was successfully attacked.²⁹ While the app has been criticized for sending messages as plaintext and therefore enabling Eve to eavesdrop, the SSL protection that was intended to secure 'sensitive' information such as registration credentials and the user's contact does not protect

²⁶appcom.zoner.android.antivirus – honored as the "Best free anti-virus program for Android" with a detection rate > 90% – <http://www.av-test.org/en/tests/android/>

²⁷com.box.android

²⁸com.yahoo.mobile.client.android.flickr

²⁹com.whatsapp

from Mallory. For instance, one is able to obtain all telephone numbers from a user's address book using a MITMA. At the time of writing, the app had an install base of 10 to 50 million users.

SSL Stripping

SSL Stripping (cf. Section 5.2.2) can occur if a browsing session begins using HTTP and switches to HTTPS via a link or a redirect. This is commonly used to go to a secure login page from an insecure landing page. The technique is mainly an issue for Android browser apps, but it can also affect other apps using Android's `webkit.WebView` that do not start a browsing session with a HTTPS site. The `webkit.WebView` was found in 11,038 apps. Two noteworthy candidates from this category concern a social networking app³⁰ and an online portal app³¹ client app. Both apps use the `webkit view` to enhance either the social networking experience or surf the portal and have 1.5 to 6 million installs. The two apps start the connection with a HTTP landing page, and one could rewrite the HTTPS redirects to HTTP and thus catch the login credentials for Facebook, Yahoo and Google.

One way to overcome this kind of vulnerability is to force the use of HTTPS, as proposed by the HTTP Strict Transport Security IETF Draft³², or using a tool such as HTTPS-Everywhere.³³ However, these options currently do not exist for Android. Android's default browser as well as available alternatives such as Chrome, Firefox, Opera or the Dolphin Browser do not provide HTTPS-Everywhere-like features out of the box, nor could any add-ons for such a feature be found.

Lazy SSL Use

Although the Android SDK does not support SSL pinning out of the box, Android apps can also take advantage of the fact that they can customize the way SSL validation is implemented. Unlike general purpose web browsers that need to be able to connect to any number of sites as ordained by the user, many Android apps focus on a limited number of hosts picked by the app developer: for example, the PayPal app's main interaction is with `paypal.com` and its sister sites. In such a case, it would be feasible to implement SSL pinning, either selecting the small number of CAs actually used to sign the sites or even pin the precise certificates. This prevents rogue or compromised CAs from mounting MITM attacks against the app. To implement SSL pinning, an app can use its own `KeyStore` of trusted root CA certificates or implement a `TrustManager` that only trusts specific public key fingerprints.

To investigate the usage of SSL pinning, 20 high profile apps were cherry-picked that were not prone to the previous MITM attacks and manually audited. An own root CA certificate was installed on the phone and a SSL MITM proxy set up that automatically created CA-signed certificates for the hosts an app connects to. Then MITM attacks were executed against the apps. Table 5.3 shows the results. Only 2 of the apps make use of SSL pinning and thus were safe from the attack. All other apps trust all root

³⁰`com.jmt.application.facebookthemes.activity`

³¹`com.yahoo.mobile.client.android.yahoo`

³²<https://bit.ly/IJrVh5>

³³<https://www.eff.org/https-everywhere>

5 Security Vulnerability Analysis of Mobile Apps

CA signatures, as long as they are part of Android's trust anchors, and thus were vulnerable to the executed attack.

Table 5.3: Results of the SSL pinning analysis.

App	Installs	SSL Pinning
Amazon MP3	10-50 million	
Chrome	0.5-1 million	
Dolphin Browser HD	10-50 million	
Dropbox	10-50 million	
Ebay	10-50 million	
Expedia Bookings	0.5-1 million	
Facebook Messenger	10-50 million	
Facebook	100-500 million	
Foursquare	5-10 million	
GMail	100-500 million	
Google Play Market	All Phones	
Google+	10-50 million	
Hotmail	5-10 million	
Instagram	5-10 million	
OfficeSuite Pro 6	1-5 million	
PayPal	1-5 million	
Twitter	50-100 million	✓
Voxer Walkie Talkie	10-50 million	✓
Yahoo! Messenger	10-50 million	
Yahoo! Mail	10-50 million	

Missing Feedback

When an app accesses the Internet and sends or receives data, the Android OS does not provide any visual feedback to the user whether or not the underlying communication channel is secure. The apps are also not required to signal this themselves and there is nothing stopping an app from displaying wrong, misguided or simply no information. Several apps were found that provided SSL options in their settings or displayed visual security indicators, however failed to establish secure SSL channels for different reasons.

Banking apps³⁴ were found in this category that could not be fully tested, since bank accounts were required there. However, these apps stated that they were using SSL-secured connections and displayed green visual security indicators, but suffered from one of the MITMA vulnerabilities shown above. It was therefore possible to intercept login credentials, which would enable one to disable cards and gather account information using the app.

Several prominent mail apps were found that had issues with missing feedback. Both were dedicated apps for specific online services. The first app³⁵ with an install

³⁴com.vrm.hessenland, com.vrm.mindenerland

³⁵com.yahoo.mobile.client.android.mail.apk

base between 10 and 50 million users handled registration and login via a secure SSL connection, but the default settings for sending and receiving email are set to HTTP. This can be changed by the user, but the user must be aware of the issue to do this and there was no indication that the emails were not protected.

An instant messaging app³⁶, with an install base of 100,000 to 500,000 users, transfers login credentials via a non-SSL protected channel. Although the user's password is transferred in encrypted form, it does not vary between log-ins, so Eve can record the password and could use it in a replay attack to hijack the user's account.

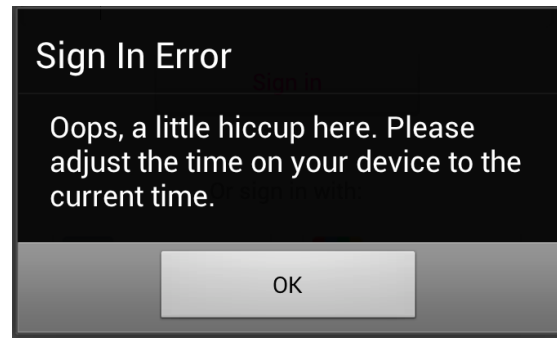


Figure 5.2: A sample warning message that occurs in an app that is MITM attacked.

A framework³⁷ was found that provides a graphical app builder, allowing users to easily create apps for Android and other mobile platforms. Apps created with this framework can load code from remote servers by using the `dalvik.system.DexClassLoader`. Downloading remote code is handled via plain HTTP. One app³⁸ built with the framework was analyzed and it was possible to inject and execute arbitrary Java code, since the downloaded code is not verified before execution.

During manual analysis, it was also found that 53 apps that were not vulnerable to the MITM attacks did not display a meaningful warning messages to the user under attack. These apps simply refused to work and mostly stated that there were technical or connectivity problems and advised the user to try to reconnect later. There was also an app³⁹ that recommended an app-update to eliminate the network connection errors. Some apps simply crashed without any announcement. Figure 5.2 shows a confusing sample error message displayed during a MITMA.⁴⁰

An additional 6 apps not vulnerable to the MITM attacks did display certificate related warning messages, but did not indicate the potential presence of a MITMA. The official Facebook app⁴¹ is not vulnerable to the MITM attacks described above and is a positive example for displaying a meaningful warning message. Even if the warning message contains tech-savvy wording, the user at least has the chance to realize that a MITM attack might be occurring (cf. Fig. 5.3).

Interestingly – excluding browser apps – there was only one app that let the user choose to continue in the presence of an SSL error.

³⁶cn.msn.messenger

³⁷<http://ibuildapp.com/>

³⁸com.appbuilder.u36633p92811

³⁹de.aboalarm.kuendigungsmaschine

⁴⁰com.yahoo.mobile.client.android.flickr

⁴¹com.facebook.katana.apk

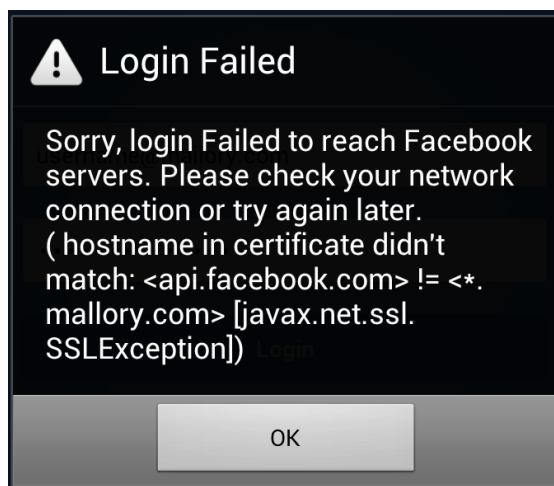


Figure 5.3: Facebook's SSL warning.

A complete list of all apps found vulnerable to on MITM attack or another can be found at the end of this section (Sec 5.2.10).

5.2.6 Limitations of our Analysis

This study has the following limitations: a) During static code analysis, the studied applications were selected with a bias towards popular apps; b) The provided install base numbers are only approximate values as provided by Google's Play Market; c) Only 100 of the apps where MalloDroid found occurrences of broken SSL implementations were manually audited. For the rest, the existence of the unsafe code does not mean that these apps must be vulnerable to a MITM attack; d) Static code analysis might have failed in some apps, for instance if they were obfuscated. Hence, there might be further vulnerable apps that were not classified as such; e) During manual audits, the applications were selected with a bias towards popularity and assumed sensitivity of data they handle; f) It was not possible to test the entire workflow of all apps, e. g., it was not possible to create a foreign bank account to see what happens after successfully logging into the bank account.

5.2.7 Trouble in Paradise

The default Android browser uses sensible trust managers and host name verifiers. Also, unlike most special purpose apps, it displays a meaningful error message when faced with an incorrect certificate and allows the user to continue on to the site if the user wants to. Thus, it relies on the ability of the user to understand what the displayed warning messages mean and what the safest behavior is. There have been many studies of this issue conducted in the context of desktop browsing. Here, to the best of knowledge, a first survey was conducted for this research to investigate the users' perceptions when using secure connections in the Android browser.

Online Survey

The goal of the online survey was to explore whether or not the user can assess the security of a connection in the Android browser. Goal was to test that a) a user can distinguish a HTTPS connection from a regular HTTP connection and b) how the user perceives a SSL warning message. Previous work has addressed the effectiveness of warning dialogues in several scenarios, mostly for phishing on regular computers (e. g., [176], [177]). Recently, Porter Felt et al. [160] conducted a survey on the prompts informing users of the requested permissions of Android apps during installation. The online survey in this research is based on a similar design, but studies SSL certificate warnings and visual security indicators in Android's default browser.

Participants were recruited through mailing lists of several universities, companies and governmental agencies. The study invitation offered a chance to win a 600\$ voucher from Amazon for participation in an online survey about Android smartphone usage. The survey could only be accessed directly from an Android phone. The survey was served via HTTPS for one half of the participants and via HTTP for the other. After accessing a landing page, the participants were shown a typical Android certificate warning message, mimicking the behavior of the Android browser. Subsequently, it was asked whether the participants had seen this warning before, if they had completely read its text and how much risk they felt they are warned against. Also of interest was to know whether or not they believed to be using a secure connection and their reasons for this belief. Finally, demographic information on technical experience, Android usage, previous experience with compromised credentials or accounts as well as age, gender and occupation was collected.

Results

754 participants completed the survey. The average age was 24 years ($sd = 4.01$), 88.3% were students while the rest mainly were employees. 61.9% of the participants did not have an IT-related education or job (non-IT experts in the following) and 23.2% had previous experience with compromised credentials or accounts. Overall, the self-reported technical confidence was high: participants stated a mean value of 4.36 for IT experts and 3.58 for non-experts on a scale from 1 (often asking for help) to 5 (often providing help to others). 51.9% of IT experts and 32.8% of non-IT experts have been using an Android smartphone for more than a year and 57.1% of experts and 69.8% of non-experts had only 25 apps or less installed.

Concerning connection security, it was found that 47.5% of non-IT experts believed to be using a secure connection, while the survey was served over HTTP. On top of that, even 34.7% of participants with prior IT education thought that they were using a secure channel when they were not. In both groups, 22.4% were unsure about the protection of their connection. Only 58.9% of experts and 44.3% of non-experts correctly identified that they were using a secure or insecure connection when prompted. The majority of users referred to the URL prefix as the reason for their beliefs and 66.5% of participants that were unsure said that they did not know how to judge the connection security. Those users that were wrongly assuming a secure connection stated that they use a trustworthy provider (47.7%), trust their phone (22.7%) or thought that the address was beginning with https:// even though it was not (21.6%) as a justification for their beliefs. Interestingly, participants that stated that they had suffered from compromised

5 Security Vulnerability Analysis of Mobile Apps

credentials or online accounts before did significantly better in judging the connection state ($\chi^2 = 85.36, df = 6, p < 0.05$).

Concerning the warning message, the majority of participants stated that they had not seen such a certificate warning before (57.6% of non-IT experts and 52.3% of IT experts) or were unsure (5.9%/9.2%). 24.0% of all participants only read the warning partially and 4.5% did not read it at all. These numbers did not differ significantly based on whether or not they had seen the warning before. The participants rated the risk they were warned against with 2.86 ($sd = .94$), with 1 being a very low risk and 5 a very high risk. The perceived risk did not differ significantly between IT-experts and other users.

Overall, the results of our online survey show that assessing the security of a browser session on Android's default browser was problematic for a large number of our participants. While certificate handling is done correctly by the browser app and basic visual security indicators are offered, the user's awareness for whether or not his data is effectively protected is frequently incomplete.

Limitations

The survey is limited in the following ways: Official mailing lists were used to distribute the invitation for the survey. While, on a technical level, this should not affect the trustworthiness of the mail or the survey site - the emails were not digitally signed and the survey was served with an URL that was not obviously linked to the university. Therefore, the emails could have been spoofed. Nonetheless, it is likely that a higher level of trust was induced in most participants, due to the fact that the survey was advertised as a university study (c.f. [178]). Therefore it was refrained from evaluating the users' reasons for accepting or rejecting a certificate in this concrete scenario.

Participants were self-recruited from multiple sources, but mainly entries from university students for this first exploration were received. While a study by Sotirakopoulos et al. [179] found little differences between groups of students and the broader population in the usable security context, a more varied sample of participants would improve the general applicability of the results.

5.2.8 Countermeasures

There are several ways to minimize the problem of unencrypted traffic or SSL misuse. They can be categorized into three groups: (1) solutions integrated into the Android OS, (2) solutions integrated into app markets and (3) standalone solutions.

OS Solutions

Enforced Certificate Checking A radical solution to prevent the use of overly permissive TrustManagers, SSLSocketFactorys and AllowAllHostnameVerifiers is to disallow custom SSL handling completely. This can be achieved by forcing developers to use the standard library implementations provided by Android's APIs. By limiting the way TrustManagers, SSLSocketFactorys and HostnameVerifiers can be used, most cases of faulty code and unintended security flaws could be avoided.

HTTPS Everywhere A solution to improve a fair number of the vulnerabilities discovered in the test set would be an Android version of HTTPS-Everywhere, integrated into the communication APIs. This would prevent most SSL Stripping attacks were found in the test set.

Improved Permissions and Policies Instead of simply having a general permission for Internet access, a more fine-grained policy model could allow for more control (cf. [159]). By introducing separate permissions for INTERNET_SSL and INTERNET_PLAIN, apps could indicate which type of connections are used. This would give users a chance to avoid applications that do not use HTTPS at all. However, in mixed mode cases or when SSL is used but used incorrectly, this method would not protect the user without additional indicators/countermeasures. Furthermore, introducing policies like GSM_ONLY, NO_OPEN_WIFI or TRUSTED_NETWORKS could help to protect apps from some MITM attacks. Despite the fact that cellular networks such as GSM/3G/4G do not provide absolute security, they still require considerably more effort to execute an active MITMA. Apps could then specify which types of networks or even which connections specifically are allowed to be used. However, this countermeasure could have considerable usability and acceptance issues.

Visual Security Feedback Reasonable feedback to the user about the security status of the currently running application is undoubtedly a valuable countermeasure – at least for some users. The operating system should provide visual feedback on whether or not apps are communicating via a secure channel. Current mobile devices usually only show the signal strength, the connection type and whether any transfers are in progress at all. Finding an effective way to inform users about which apps are currently communicating with the Internet and whether the communication is secure is not trivial and should be studied carefully before a solution is propagated.

MalloDroid Installation Protection MalloDroid could be integrated into app installers, such as Kirin [158], to perform static code analysis at install time. This analysis performed directly on a phone could warn of potentially unsafe applications. The user would then have to decide if he wishes to install the app irrespective of the warning.

App Market Solutions

Similar to the MalloDroid installation protection, MalloDroid could be integrated into app markets. This form of automated checking of apps could either be used to reject apps from entering the market or warnings could be added to the app's description. Both options have usability and acceptance issues that need to be studied.

Standalone Solution: The MalloDroid App & Service

All countermeasures mentioned above require modification of the Android OS and support from Vendors and/or app markets. Standalone solutions can be deployed more easily. Therefore, as a stop-gap measure, the MalloDroid tool is going to be offered both as a Web and Android app. This will at least allow interested users to check on apps before they install them. The Android app will also offer a convenience feature checking all installed applications. MalloDroid can of course also be used as-is with Androguard.

5.2.9 Conclusion

In this research, an investigation of the current state of SSL/TLS usage in Android and the security threats posed by benign Android apps that communicate over the Internet using SSL/TLS was presented. MalloDroid, a tool that uses static code analysis to detect apps that potentially use SSL/TLS inadequately or incorrectly and thus are potentially vulnerable to MITM attacks, was developed. The analysis of 13,500 popular free apps from the Google Play Market has shown that 1,074 apps contain code belonging to this category. These 1,074 apps represent 17.0% of the apps that contain HTTPS URLs. To evaluate the real threat of such potential vulnerabilities, MITM attacks against 100 selected apps from that set have been mounted manually. This manual audit has revealed wide spread and serious vulnerabilities. The credentials for American Express, Diners Club, Paypal, Facebook, Twitter, Google, Yahoo, Microsoft Live ID, Box, WordPress, IBM Sametime, remote servers, bank accounts and email accounts have been captured. It was possible to successfully manipulate virus signatures downloaded via the automatic update functionality of an anti-virus app to neutralize the protection or even to remove arbitrary apps, including the anti-virus program itself. Furthermore, it was possible to remotely inject and execute code in an app created by a vulnerable app building framework. The cumulative number of installs of apps with confirmed vulnerabilities against MITM attacks is between 39.5 and 185 million users, according to Google's Play Market.

The results of the online survey with 754 participants showed that there is some confusion among Android users as to which security indicators are indicative of a secure connection, and about half of the participants could not judge the security state of a browser session correctly. Possible countermeasures were discussed that could alleviate the problems of unencrypted traffic and SSL misuse. MalloDroid is offered as a first countermeasure to possibly identify potentially vulnerable apps.

The findings of the investigation suggest several areas of future work. The intention is to provide a MalloDroid Web App and Android App and will make it available to Android users. Moreover, there seems to be a need for more education and simpler tools to enable easy and secure development of Android apps. But most importantly, research is needed to study which countermeasures offer the right combination of usability for developers and users, security benefits and economic incentives to be deployed on a large scale.

5.2.10 List of Apps With Broken SSL Usage

The following table gives an overview of the apps that we found vulnerable to MITM attacks. Either accepting all certificates (1) or allowing all hostnames (2) or being vulnerable for SSL Stripping (3).

App	Installs	Vuln	Comment
Bankdroid	100k-500k	(1)	Banking app. Leaks banks, Mastercard, Diners Club and Paypal credentials.
Börse Mobil	10k-50k	(1)	Shows stock market information. Leaks login credentials.
CashBase	1k-5k	(1)	Wallet app, keeps track of spendings. Leaks login credentials.
Dolphin Browser HD	10m-50m	(1)	Web browser. Leaks credentials for Dolphin Connect.
FriendScout24	10m-50m	(1)	Dating App. Leaks login credentials.
IBM Sametime	10m-50m	(1)	IBM Sametime client. Leaks login credentials.
IP Cam Viewer	100m-500m	(1)	App for viewing and controlling IP cameras. Leaks login credentials, gives Mallory access video-/audiostream.
Last FM	1m-5m	(1)	App for sharing favorite music. Leaks data during login process.
Mamba	100k-500k	(1)	Dating app. Leaks account information.
Messenger With You	10m-50m	(1)	MSN Messenger. Sends the credentials in an obfuscated way.
Mobli	100k-500k	(1)	Photo sharing app. Leaks credentials, contacts and GPS location.
Pinger SMS Free	10k-50k	(1)	SMS app. Leaks login credentials.
Pizza.de	100k-500k	(1)	Pizza ordering app. Leaks login credentials.
SonicWALL Mobile Connect	5k-10k	(1)	Firewall administration, Establishes connection via broken SSL.
Total Recall	500k-1m	(1)	Call recorder. Checks premium status over broken SSL.
TouchDown Tablet	50k-100k	(1)	Exchange and Active Sync mail app. Leaks login credentials.
VR Banking	1k-5k	(1)	Banking application. Credentials and PIN are transmitted over broken SSL.
VZ-Netzwerke	500k-1m	(1)	Social network client. Leaks data during login process.
WordPress	1m-5m	(1)	Blogging tool. Leaks account information.
World of Tanks Assistant	100k-500k	(1)	Gaming assistant tool. Leaks credentials.
xScope Browser	500k-1m	(1)	Browser, that accepts arbitrary certificates.
AlwaysOnPC-HD:Office,Chrome	10k-50k	(2)	Virtual PC app. Leaks login credentials.
BILD	500k-1m	(2)	News app with premium features. Checks premium subscription over broken SSL, allowing a MITM to obtain such a subscription.
Box	1m-5m	(2)	Document sharing application for enterprises. Leaks login credentials.

5 Security Vulnerability Analysis of Mobile Apps

Cisco WebEx Meetings	100k-500k	(2)	Cisco WebEx client. Leaks login credentials.
DS file	100k-500k	(2)	Access remote files.
Flickr	500k-1m	(2)	Flickr client. Leaks Facebook/Google credentials during login.
Forfone	500k-1m	(2)	App for the SIP protocol and text messages. Leaks login credentials and data.
Formspring	100k-500k	(2)	Questionnaires and photo sharing app. Leaks login credentials.
Hipster	10k-50k	(2)	Photo sharing and social network app. Leaks login credentials.
HootSuite	100k-500k	(2)	Social networking app for Facebook, Twitter, etc. Leaks login credentials.
HP iLO Mobile	5k-10k	(2)	Control remote servers. Allows to shutdown servers. Leaks login credentials.
MailDroid	500k-1m	(2)	MailDroid is a free IMAP/POP3/Exchange email client. It leaks login credentials as well as emails themselves.
Miner Status	5k-10k	(2)	Bitcoin mining status app. Leaks the miner-API key.
Monster	1m-5m	(2)	Job search app. Leaks login credentials.
Moxier Mail Trial	100k-500k	(2)	Exchange Active Sync mailing app. Leaks login credentials.
Posterous Spaces	100k-500k	(2)	Photo and Video sharing app. Leaks login credentials.
Privat24	50k-100k	(2)	Banking app. Leaks login credentials.
TouchDown Phone	500k-1m	(2)	A free Exchange and ActiveSync client. Leaks login credentials.
WhatsApp Messenger	10m-50m	(2)	Messaging app. Leaks credentials during registration.
Zoner AntiVirus Free	100k-500k	(2)	Antivirus App. Loads signatures over broken SSL, allows the injection of arbitrary signatures.
Facebook Themes	500k-1m	(3)	Color theming for the Facebook website. Leaks Facebook credentials.
LetMeCU	100k-500k	(3)	Dating app. Leaks login credentials.
QuickWindowsLiveHotr	100k-500k	(3)	Email app. Leaks login credentials.
Yahoo!	1m-5m	(3)	Yahoo! news and services. Leaks login credentials.

5.3 AndroLyze: Static Mobile App Analysis

5.3.1 Introduction

With the rise of mobile applications (“*mobile apps*”) and the availability of app market places, the security and privacy risks associated with mobile apps increase accordingly. Not only mobile malware is growing, but also the risk for private and corporate data posed by badly written software as shown in the previous section. As a consequence, more and more security checks are developed [159] [9] [180] [181] [182] to analyze mobile apps. The focus is not solely on identifying malicious software, but also on spotting potential privacy breaches [183] and corporate data leaks [184] or indicating bad programming issues such as energy bugs [185]. In the last years, researchers did not only look at cherry picked applications from various market places, but also mass-analyzed apps ranging from a few hundred [159] over several thousand [9] up to over a million mobile apps [182].

Performing mobile app analysis in an automated manner is interesting for the research community as well as for the corporate world. However, setting up an environment for different analyses is a tedious task, and conducting mass security scans of apps is quite time-consuming.

In this section, AndroLyze, a distributed framework to analyze large numbers of apps in an efficient manner, is presented. AndroLyze combines several key features in a novel way that distinguish it from other work on mobile app analysis:

- AndroLyze offers a utility library for standardized writing of static analysis scripts
- AndroLyze provides unified logging and reporting functionality backed by a database
- AndroLyze can handle large sets of mobile apps, including different versions of an app
- AndroLyze achieves efficiency through parallelization and distribution among CPU cores, CPUs and servers
- AndroLyze relies on optimized job scheduling to obtain faster analysis times

By having a standardized way of writing and deploying scripts as well as a properly defined output format, it is much easier to incorporate security checks into the development process or use the knowledge of various script sources to improve corporate security by enriching black-/white-lists of mobile device management solutions. To demonstrate the benefits of AndroLyze, the Top Free 500 Android apps of all categories in *Google Play* collected over three years are analyzed. The whole data set consists of almost 40,000 apps requiring about 227 GB of storage space.

Parts of this section have been published in [10].

5.3.2 Related Work

Several approaches in the literature rely on Android app analysis, ranging from the detection of privacy leaks to misuse of cryptography and certificates as well as identification of malicious software. However, very few publications address the issue of mass-analysis of Android apps. Most approaches are “hand-made” and are only suitable for a particular use case, and they do not provide version and result management.

A common basis for manual and automatic app audits has been developed by Desnos et al. [186]. This framework written in *Python* is called *Androguard* and offers features such as disassembly, decompilation, control flow graphs and similarity search. The high number of features and the possibility to easily write scripts using them, combined with constant improvements over the years, added to *Androguard*'s popularity.

Viennot et al. [182] have conducted a study of *Google Play* and developed *PlayDrone*, a crawler of the *Google Play* store including a distributed analysis framework. Downloaded APKs are decompiled with *dex2jar* and *JD-Core* and stored in *Git* repositories located on worker nodes as well as on *ElasticSearch*. The latter offers full text search and an analysis engine for an app's source code. *AndroLyze* differs from *PlayDrone* in the fact that it offers a native decompiler and uses *MongoDB*, a *NoSQL* database for result evaluation and storage. Furthermore, it achieves efficiency through parallelization and optimized job scheduling, and can handle different versions of an app.

Fahl et al. [9] have conducted a mass audit of APKs using a script based on *Androguard* named *MalloDroid* to identify security issues in the use of transport layer encryption in apps. No special infrastructure used in the analysis is presented, and the system is specifically tailored to perform SSL/TLS analysis. Cryptography related analysis has been performed by Egele et al. [187] in their empirical study. Their static analysis tool called *CryptoLint* is also based on *Androguard*, and the number of APKs analyzed is comparable to the work by Fahl et al. [9]. A similar approach has been taken by Felt et al. [159] for their static analysis tool *Stowaway* to analyze app permissions. Using *Dedexer* to disassemble the APKs, the output is analyzed and the results are manually stored. Further research on permission based problems with Android applications has been done by Bartel et al. [188] by using static analysis to build call graphs, but focusing on the Android framework itself and not on the apps. Fend et al. [189] use static analysis of applications to detect malware. By performing inter-component control flow analysis and static taint analysis, APKs from *Google Play* are analyzed. Static analysis is based on *SOOT*⁴² for the task at hand. Kim et al. [183] have developed static tests to detect privacy leaks in Android applications. The main component is a modified decompiler and an intermediate language for their analyzer. Another approach to static analysis has been presented by Payet et al. [190] by extending the *Julia* static analyzer to work with the specific requirements and features of Android applications. Only tens of apps have been tested, and with a runtime of 7 minutes per app, speed can be an issue for mass audits. Static analysis is also part of the work done by Schmidt et al. [191]. For their collaborative malware detection function, calls made in the Android environment are extracted for automatic classification and comparison with malware. All of these papers focus on their specific tests and have their own implementation to handle large amounts of APKs. In contrast, *AndroLyze* provides a standardized way offering a single platform for all kinds of security checks regardless of the target and also with analysis speed in mind to execute them as fast as possible.

Grace et al. [181] have developed *RiskRanker* to detect apps exhibiting dangerous behavior. Apart from offering this particular type of analysis, the authors focus on execution speed to potentially handle large numbers of APKs. Nevertheless, *RiskRanker* does not provide a general purpose platform for other tests or a mixture of different analysis scripts.

SAAF developed by Hoffmann et al. [192] is a general framework for static Android

⁴²<https://sable.github.io/soot/>

app analysis. It can be used for manual and automated checks. While it aids developers in writing new security checks and provides fast execution times for single scripts, it does not provide support for effectively handling results and large numbers of APKs and scripts.

An approach combining different solutions for Android malware detection has been presented by Maggi et al. [180]. Similar to Google's VirusTotal⁴³, *AndroTotal* is a platform based on virtual Android devices running different malware detection software and giving users submitting samples feedback based on the results. The focus is solely on an infrastructure for dynamic analysis including user-interaction automation. Integration of different ready-to-run native anti-malware apps in the Android emulators is provided, but there is no support for third-party developers, and there is no scheduler optimized for fast results while conducting mass audits in place.

Andrubis developed by Weichselbaum et al. [193] is a platform combining dynamic and static analysis. Static analysis is used to aid and fine-tune dynamic analysis and providing additional information about an app, such as permissions, activities, broadcast and receivers. While *Andrubis* has been designed for broad tests written for dynamic analysis on larger sets of APKs, the requirements for this hybrid approach are quite different from AndroLyze, and it lacks the unification and rapid writing of static analysis tasks AndroLyze provides.

5.3.3 AndroLyze's Design

This subsection presents the design and analysis approach taken by *AndroLyze*, as illustrated in Figure 5.4. The bottom part of Figure 5.4 shows that AndroLyze has a direct link to *Google Play* to download apps. Moreover, it shows an analyst with an import database carrying information about the APKs to be analyzed. Importing apps into a local database enables certain filter and sorting capabilities, but AndroLyze can also be used without using a local database. AndroLyze already comes with a built-in set of scripts, such as extracting permissions from the manifest, disassembling or decompiling the APK. Additional analysis criteria require users to write custom scripts based on the functionality of *Androguard* [186].

The top part of Figure 5.4 illustrates the analysis approach. There are different modes: Depending on the number of applications, a user can start an analysis either locally (local mode) or on a cluster of nodes (distributed mode) to further improve performance. Both modes operate in a fully parallel manner, leveraging all available cores of the used processor(s).

The distributed mode shown in Figure 5.4 works as follows: Initially, the scripts are deployed via *SSH* on the available nodes. Tasks are sent to a distributed job queue and processed by a pool of worker nodes. The APKs are either stored in the jobs or pre-distributed and available in the APK cache. After a node has finished a job, it stores the outcomes in the result database. Finally, an analyst can view and evaluate his or her findings either using *AndroLyze* directly, sending custom queries to the database, or by syncing all results and performing a local analysis or review.

AndroLyze is designed to analyze large numbers of APKs in a short amount of time. More nodes can be added on the fly to further improve performance and reduce the analysis time.

⁴³<https://www.virustotal.com>

5 Security Vulnerability Analysis of Mobile Apps

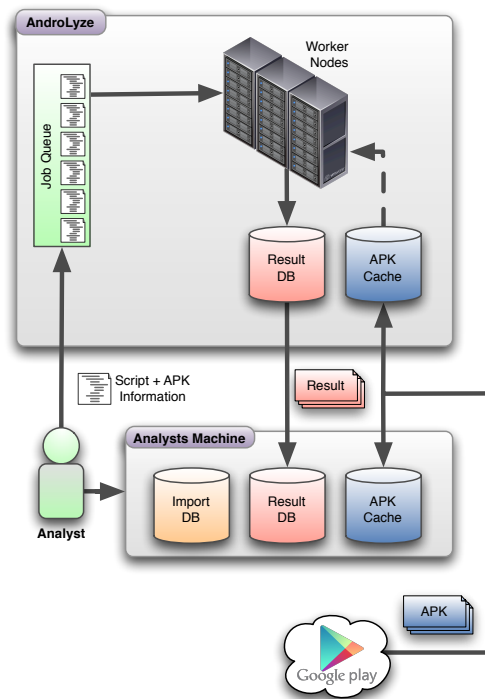


Figure 5.4: Analysis approach of *AndroLyze*

In the following, the design of *AndroLyze* is explained in more detail, describing the script framework, the storage system and the parallel analysis engine. Finally, the reader is guided through a typical workflow using *AndroLyze*.

Script Framework

The heart of *AndroLyze* is the script framework. User defined analysis features can be implemented with Python scripts leveraging the power of *Androguard* [186].

Script Requirements Scripts in *AndroLyze* can be written for different purposes, such as collecting information from the Android manifest, accessing information gathered by analyzing the disassembly, checking where and which permissions are used in the code, creating the control flow graph or decompiling the application. These tasks differ in their demands they pose on *Androguard*. *AndroLyze* takes advantage of this fact to improve performance by requiring script authors to signal their requirements to implement certain analyses. Therefore, *AndroLyze* can scale down the requirements of all scripts to the lowest common denominator, resulting in the best analysis performance. The built-in scripts are written in a modular way, and functionality can be chained together on demand.

Logging Logging the results of an analysis in a structured fashion is important for any evaluation study. The *AndroLyze* script framework comes with a set of functions to log common data types. Additionally, results can be summarized to groups of particular

interest. Examining the manifest, for instance, might lead to a structure such as *manifest* \rightarrow *permissions* \rightarrow $\{permission_1, \dots, permission_n\}$.

Very large and/or binary data is supported by the logging component. It is directly connected to the storage architecture, thus all results are kept in the result database and optionally in the file system of the analyst.

AndroLyze does not enforce a static schema for results, but requires a script author to register a basic structure such that a result can be rendered to hold initial default values. This offers the possibility to visualize results in the absence of any logging and makes the comparison of app versions easier.

Storage & Analysis

AndroLyze provides an *import database* that carries meta-information about the APKs, a *result database* for storing the results and an APK cache for the .apk files. One of the main design goals of the *import database* is to manage different versions of an application, either from different sources (e.g. *Amazon Market* or *Google Play*) or different builds. This can be used for historic analysis, e.g., to see how long different bugs were present or to determine application trends on a broader scale, such as the adoption of technologies like HTML5-based frameworks or the use of encryption over the course of several month or years. To collect these different versions, *AndroLyze* allows analysts to update the database by downloading newer versions, if present, directly from *Google Play*. Thus, evaluations can not only be performed on the most recent APKs, but also on earlier versions several years back.

The analysis can be performed in two different modes, called *local parallel* and *distributed parallel*. The local parallel mode offers the possibility to use *AndroLyze* without the need to install a distributed system. The distributed parallel mode enables the system to further improve performance by adding more nodes on demand.

Both implement *dynamic scheduling* to utilize all available nodes, hence all cores/processors as long as possible. However, since the analysis time between jobs may differ strongly, long running jobs are scheduled first. Since most scripts analyze some piece of code, the size of the *classes.dex* file is used as a simple metric for a job's runtime. This approach is called Code Size Scheduling (CSS). With both *dynamic scheduling* and CSS, parallel execution can be performed as long as possible.

The actual analysis process is as follows: A worker fetches a job from the task queue, performs an analysis and stores the outcomes in the result database. A job represents the analysis of a single APK with a given set of scripts. An APK is either part of the job or fetched from the APK cache if the applications have been imported beforehand. In addition to the analysis results, the system stores the identifiers of the analysis results in the job queue such that the analyst can view his or her findings directly after the analysis has finished.

Workflow

To illustrate the approach in action, the reader is now guided through a typical workflow while using *AndroLyze*.

Figure 5.5 summarizes the architecture and common steps required for an analysis. First, the analyst needs to obtain the APKs (s)he wants to inspect. The link to *Google*

5 Security Vulnerability Analysis of Mobile Apps

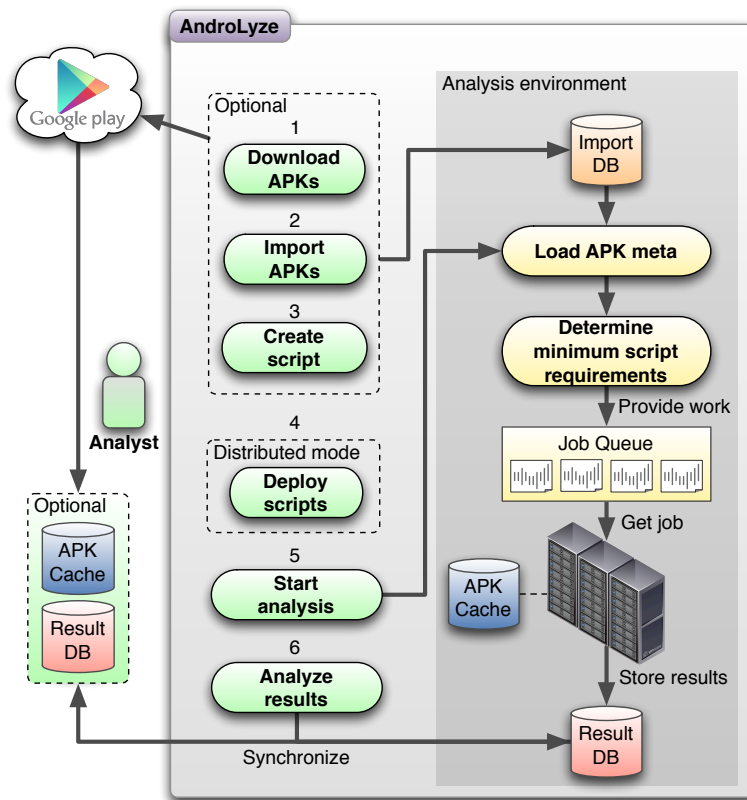


Figure 5.5: Typical workflow while using AndroLyze

Play⁴⁴ offers the possibility to download single applications via the package name or to download the newest or most famous APKs from either a particular category or all categories (Step 1). Importing the apps into a local database (Step 2) is not required, but enables an analyst to use our advanced scheduling and filtering functionality. The next step is to create a script (Step 3) suitable for AndroLyze, leveraging the power of Androguard in order to implement the needed analysis functionality. The more nodes one has, the faster the entire analysis is. Therefore, it is recommended to use the *distributed parallel* mode for larger sets of applications. It assumes that all nodes have been started and that the scripts have been deployed (Step 4). The deployment process can be handled through AndroLyze and requires an analyst to have *SSH* access on the nodes. After Steps 1-4, the actual analysis process can be triggered (Step 5). Initially, the APK meta-information is loaded, either from the .apk files supplied by the analyst or from the *import database*. For an efficient analysis (Step 6), the minimum requirements that satisfy all scripts are determined and sent together with information about the scripts and the applications to the *distributed job queue*. The worker nodes in the pool fetch one job after the other, perform the analysis and store the outcomes in the *result database*. The results can be synchronized, concurrently to the analysis process, to the local hard disk drive of the initiator of an analysis. Finally, the findings can be further processed or evaluated.

⁴⁴The listing of the most famous or newest applications in a certain category is limited by $n = 500$ [182]

5.3.4 Implementation

In this subsection, implementation details of *AndroLyze*, including the script framework, the storage and the analysis components, are described.

Script Framework

There are two types of scripts, implemented as subclasses of either *AndroScript* or *ChainedScript*. The first provides the analysis functionality for scripts, the latter enables the chaining of scripts to bundle their functionality. Depending on the script requirements the analyst needs, the following analysis objects are provided:

- **DalvikVMFormat**: Creates a disassembly and provides access to classes, fields and methods.
- **VMAalysis**: Analyzes the disassembly to check, for example, where which permissions are used or if reflection or dynamic code loading is used.
- **GVMAnalysis**: Creates the control flow graph.
- **XREF**: Detects cross-references between methods.
- **DREF**: Detects cross-references between data fields.

Through the definition of requirements, *AndroLyze* can query all scripts and determine the minimum script requirements to provide the necessary analysis functionality. Without defining any requirements, it is only possible to access the raw *.apk* file or its contents, such as the manifest file.

Every script has access to the logging framework, allowing unified logging in a structured fashion. *AndroLyze* supports logging of all *JSON* serializable data types, such as booleans, integers, strings, lists and dictionaries. Internally, a dictionary keeps all reported results, extended by static information about the analyzed APK as well as the script. Moreover, *AndroLyze* offers a custom storage interface for text files, graphs or other binary data. Statistics about the analysis time help to improve the script performance.

Storage

The storage implementation is divided into two parts: the import and the result database. Both are described below.

Import Database Even if the analyst has many APKs, (s)he might be interested in a subset or a single application only. To provide filters, *AndroLyze* needs information from the application's manifest. The import mechanism extracts data such as the *package name* and *version number* from the manifest. Moreover, it stores the size of the *classes.dex* file as a metric for the script runtime used for the improved *job scheduling*. Instead of relying on *Androguard* to perform the import, a faster solution was developed that extracts only the manifest, thus preventing the extraction of the whole *.zip* file as *Androguard* does it. The import process is fully parallelized and all operations are executed in-memory after the APK has been loaded. A *SHA 256* message digest⁴⁵ over the whole APK serves as a unique key in the database and enables management of different versions of a single app. Therefore, *AndroLyze* can provide a security track record over many releases.

⁴⁵In the following, *message digest* or *hash* always refer to the usage of the *SHA 256* algorithm

5 Security Vulnerability Analysis of Mobile Apps

Labeling a data set in order to provide a better distinction between different APK sets is also allowed. Finally, the meta-information is stored in a local *SQLite* database so that different analysts can manage their databases independently of others.

Result Storage In contrast to the local import database that is basically a *flat file*, *MongoDB*⁴⁶, a schema-free *NoSQL* database for the distributed result storage, is used. Therefore, all users of *AndroLyze* share the same result database. Nevertheless, different views on the database can be used to distinguish between data sets.

AndroLyze adds meta-information for an APK, such as package name, version name, build date and message digest, to the result. Additionally, information such as script name, hash, version number and the analysis date are added as script meta-information. This meta-information about the APK and script add fields to the result layout, allowing to formulate queries for the database without any knowledge of the actual results. A result is defined as the outcome of the analysis of a single APK using a single script. New results with the same script name and APK hash replace old ones⁴⁷. In addition to the result storage in the database, *AndroLyze* can store the results formatted as JSON strings on the local hard disk drive.

GridFS To support the storage of files larger than the *MongoDB* limit of 16 MB, *AndroLyze* uses *MongoDB*'s *GridFS* that splits *BSON* documents into binary *chunks*. Two collections, one for the *chunks*, the other for the meta-information, are used to reassemble the whole file. The method is used for storing large results and APKs⁴⁸. *AndroLyze* does not decide on the fly to use *GridFS* for large results due to the fact that storing results in a binary format allows analysts to query only the meta-information of the results.

Key Escaping Reserved characters such as "." and "\$" need special treatment if they are used as keys in the result. This is important when using the package name in a query. *AndroLyze* escapes them with "_" and "_\$", respectively. *Key escaping* is only necessary if queries are sent directly to *MongoDB*, since *AndroLyze* automatically handles this while creating queries.

Analysis

Although the local parallel and distributed parallel mode share the same conceptual design, they differ in their implementation details. Both modes use processes instead of threads, because, due to the *Global Interpreter Lock*, only one Python thread can execute code at once in the interpreter. On each node, as many processes as the CPU has cores (including multithreading) are spawned. Providing the *Androguard* analysis objects, such as the disassembly, implicates overhead that can be reduced by the definition of script requirements. This enables *AndroLyze* to use the minimum script requirements needed to perform an analysis. Due to the overhead involved, an APK is opened only once and all scripts are run afterwards.

⁴⁶<https://www.mongodb.org>

⁴⁷The keys for the results are generated by hashing the script name and the APK message digest

⁴⁸We integrated the *GooglePlayCrawler* into *AndroLyze* to access the *Play Store* and download the APKs

Local Parallel Mode The local parallel mode uses a synchronized queue to distribute the work among the processes. The queue memorizes the path to the APK and the meta-information. After a successful analysis, the results are stored in *MongoDB*.

Distributed Parallel Mode The *Celery* framework⁴⁹ is used to implement a message-oriented middleware in combination with *RabbitMQ*⁵⁰ as a distributed task queue. *Celery* acts as a message broker and provides worker instances to execute the actual analysis jobs. Each process maintains a single *Celery* task instance that is kept alive as long as *Celery* is started on the node. Therefore a persistent database connection can be kept for the result storage. To avoid idle times, a process reserves one task per process that is preloaded. Therefore, if the APK has been integrated into the message, it is already available when the process starts the analysis. The actual analysis process is as follows: Tasks are serialized with the Python *Pickle* module and sent to *RabbitMQ*. *Celery* workers retrieve a job and perform the analysis on the given APK with all scripts. Afterwards, the results are stored in *MongoDB*. The identifiers of each result are kept in *RabbitMQ* in a reserved queue only for the job results. The initiator of an analysis has a callback handler registered for this queue so that (s)he can synchronize the results concurrently to his or her local hard disk drive. All steps in the analysis process are fault tolerant. Thus, if a node fails, the job can be executed by any other node. The connection to *RabbitMQ* and *MongoDB* can be fully encrypted, using X.509 certificates.

5.3.5 Experimental Evaluation

In this subsection, the performance of *AndroLyze* is evaluated. First, speed improvements by import parallelization and by on-demand script requirements are demonstrated. Afterwards, the local parallel and distributed parallel modes are investigated as well as the APK distribution strategies. Then, the overall performance of *AndroLyze* is illustrated by evaluating the analysis time of almost 40,000 applications (i.e., the Top Free 500 APKs from all categories collected in three years: 2012, 2014 and 2015). Finally, the results of a study on the use of cryptographic code in this set of APKs is presented.

Test Environment

In total, 7 computers are used in the experiments. Each has an Intel Core i7-4771 processor, 32 GB RAM, a 240 GB SSD, and two hard disk drives with both 3 TB storage. They are connected via Gigabit Ethernet, and all except one computer host a KVM virtualized machine (VM) that has only 16 GB RAM and a 128 GB HDD (no RAID), except for the job queue which has 512 GB of additional space.

Table 5.5 shows the APK sets used in the experiments: the Top Free 4, 100 and 500 applications⁵¹ available from all categories in *Google Play*. Furthermore, snapshots of the Top Free 500 applications of the years 2012, 2014, and 2015 were collected, and merged in the set *Top Free 500 Archive* to demonstrate the versioning system and security track capabilities of *AndroLyze*.

⁴⁹<http://www.celeryproject.org>

⁵⁰<http://www.rabbitmq.com>

⁵¹The shown numbers do not take duplicate apps (same hash) into account. Duplicates can occur because an app may be present in more than one category

5 Security Vulnerability Analysis of Mobile Apps

Table 5.5: APK test sets

Set	# unique APKs	Size in MB
Top Free 4	102	1,159
Top Free 100	2,519	22,315
Top Free 500	12,689	91,764
Top Free 500 Archive	39,725	226,798

The used script sets shown in Table 5.6 start with scripts that extract information from the manifest only. The next set adds partial SSL analysis capabilities from Mallo-Droid (Sec. 5.2). *Bytecode* has additional scripts listing the classes and fields using the disassembly as well as creating a *method call graph*. In addition to *Bytecode*, *Source code* performs decompilation with the help of *Androguard's DAD decompiler*.

Table 5.6: Script test sets

Set	Reqs.	Scripts
Manifest	None	Activities, BroadcastReceivers, ChainedApkInformation, ContentProviders, Files, Intents, Libs, Permissions, Services
+SSL	XREF	<i>Manifest</i> , SSL
+Bytecode	XREF	<i>+SSL</i> , AnalyzeFrameworks, ClassDetails, ClassListing, MethodCallGraph
+Source code	XREF	<i>+Bytecode</i> , Decompile

Import

The first experiment is performed on a single computer and inspects the performance improvements achieved by parallelizing the import and using the import code compared to the standard *Androguard* import.

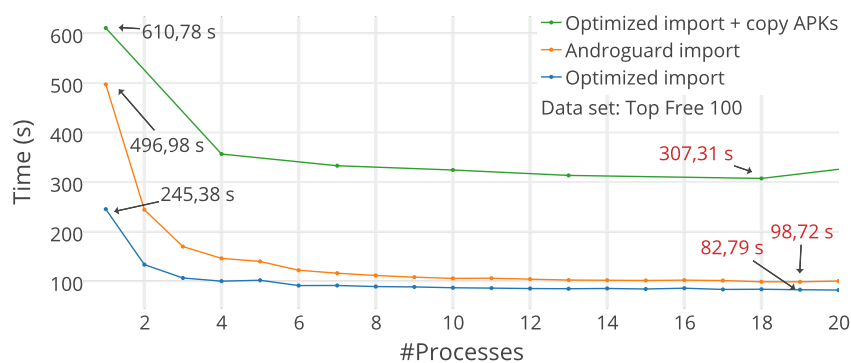


Figure 5.6: Parallelization of APK import

Figure 5.6 shows three curves: The orange (middle) curve is the import using *Androguard* to open the APK and to extract the needed manifest information. The blue (bottom) curve points out the speed improvements over *Androguard* since only the

manifest needs to be extracted from the *.zip* file and all other processing of the APK can be omitted at this stage. The green (top) curve shows the influence of concurrently copying the APKs to the local hard disk drive. For each run, the import database as well as the import directory for the APKs were deleted. The database is created on the SSD while the APKs are copied from and to the local hard disk drive. With only one process, the improved import mechanism is 2.03 times faster than *Androguard*. Even though the performance benefits decrease with more processes, there is still an improvement of 19.24% with 19 processes, which is the best value for the *Androguard* import. Moreover, copying the APKs clearly reduces the import speed because the hard disk drive is involved, but even here there is a performance increase of 98.87% with 18 processes compared to 1 process.

Scripts

The next experiment measures the impact of the analysis object provisioning time on the script runtime. For this purpose, a script was created for each available analysis object, signaling only the particular script requirement that is to be examined. The method body of the procedure responsible for the actual analysis was left empty. Nevertheless, meta-information about the script and APK is still logged and stored in the file system as well as in *MongoDB*, because this is automatically done by *AndroLyze*. The network is not involved in this test scenario, since only a single computer with a local *MongoDB* instance in the *local parallel* mode is used.

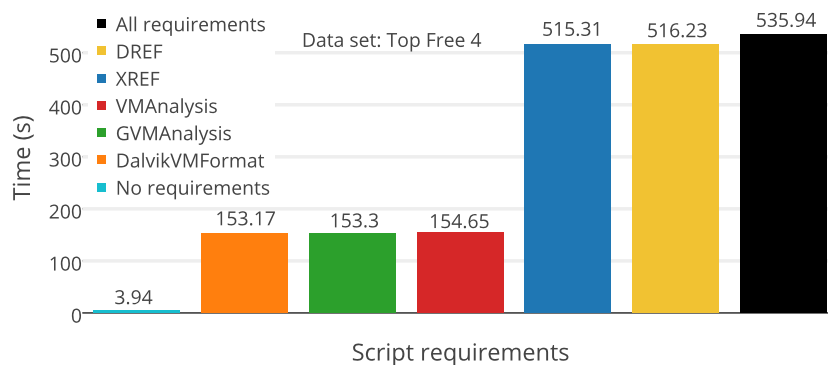


Figure 5.7: Relation between script requirements and runtime

The results of this experiment shown in Figure 5.7 indicate that providing the analysis objects only on demand improves performance. Providing access to the manifest for all 102 applications of the APK set *Top Free 4* is very fast. As soon as the first requirement is needed, the runtime gets approximately 100 seconds longer (DalvikVMFormat, VMAnalysis and GVMAnalysis) because the disassembly etc. have to be created. Creating cross-references between methods (XREF) and fields (DREF) are the most time-consuming requirements.

Local Parallel Mode

To evaluate the performance of the *local parallel* mode, the APK set *Top Free 4* was used and then analyzed with the script set *Source code*. The results in Figure 5.8 show that a

5 Security Vulnerability Analysis of Mobile Apps

speed-up of 3.19 is achieved using 4 processes instead of only 1 process, each of them assigned to a single CPU core of the used computer that is equipped with 4 CPU cores.

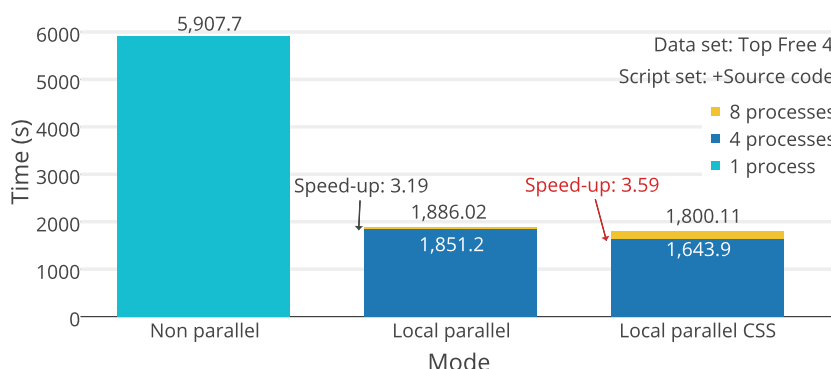


Figure 5.8: Local parallel mode +Source code

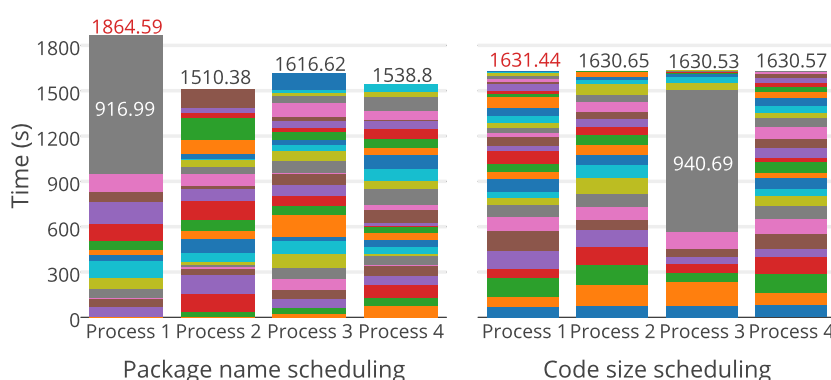


Figure 5.9: Comparison of job scheduling strategies

Figure 5.9 shows the distribution of analysis tasks among the CPU cores using two different scheduling strategies. The first scheduling strategy is based on the ascending order of the package names, while the second (CSS) uses the code size and schedules jobs with a large code sizes first. Each job is represented by a different color and its block size in the chart corresponds to its runtime. The figure shows that the analysis of a particular job took very long, which is caused by the decompilation script. With CSS, this long running job is scheduled earlier, resulting in a reduced analysis time (233.15 seconds). With improved scheduling, the speed-up of 3.19 shown in Figure 5.8 is improved to 3.59.

To validate the performance benefits of CSS, an experiment was carried out that uses the same script set as the experiment before, but does not use the decompilation script, thus eliminating the long running process. The results in Figure 5.10 show again that CSS exhibits a better performance. The speed-up is improved from 3.69 to 3.8.

Distributed Parallel Mode

To evaluate the *distributed parallel* mode, the performance properties of both modes including CSS are compared. Using 4 computers (3 of them with a KVM VM each), the distributed environment has been set up as follows: The result database is located on

5.3 AndroLyze: Static Mobile App Analysis

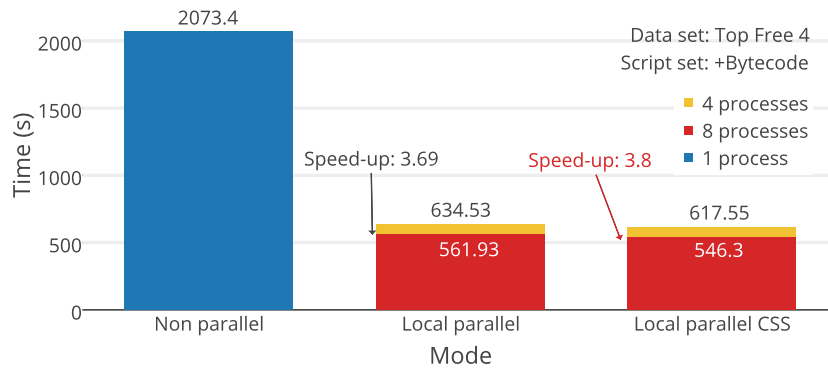


Figure 5.10: Local parallel mode +Bytecode

the computer without a VM, whereas the distributed message queue runs in a VM. The analyst initiates the analysis from the computer without a VM and integrates the APKs located on the hard disk drive into the messages sent to *RabbitMQ*. The serialization of the APK data as well as the network is taken into account.

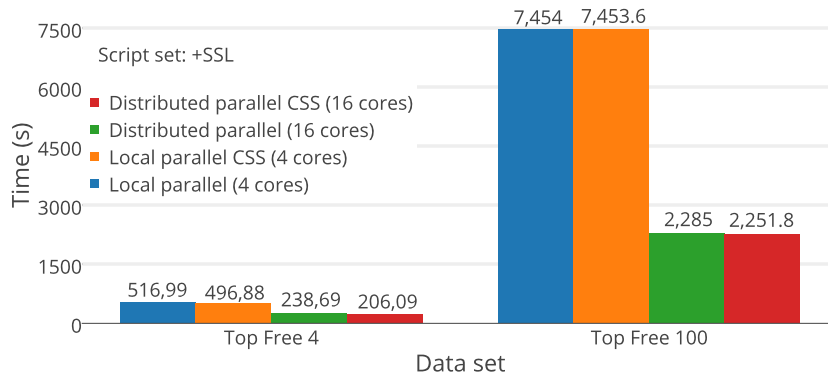


Figure 5.11: Distributed parallel mode + SSL vs. local parallel mode +SSL

Figure 5.11 shows the analysis of the data set Top Free 4 (left) and the analysis with the set Top Free 100 (right). Both use the script set +SSL, and each physical machine runs 8 processes. The results for Top Free 4 indicate that *AndroLyze* is optimized for a large number of jobs. The speed-up between the *local parallel* and *distributed parallel* mode is only 2.17 and 2.41 with CSS. This result can be explained by the reservation of tasks. Each process reserves one task, hence in the worst case, a worker may be free for work, but all other workers have reserved the remaining tasks. Another problem is the granularity of the tasks, because in our scenario the last 31 jobs are executed by less than 32 processes (the total number of processes). The results for Top Free 100 show that a speed-up of 3.26 and 3.31 with CSS could be obtained. In both cases (Top Free 4 and Top Free 100), CSS improves the performance.

APK Distribution

The next experiment investigates the data throughput of the proposed APK distribution strategies. Figure 5.12 compares both APK distribution strategies (Send APK vs. Send APK-ID) in the distributed parallel mode with the local sequential and local parallel

5 Security Vulnerability Analysis of Mobile Apps

mode. The used test set *Top Free 500* consists of 12,689 APKs with a total size of 91.8 GB to measure the throughput of serializing the APKs and the distribution through *MongoDB*. The script set *Manifest* for unzipping the APK and extracting the information, is used in this experiment.

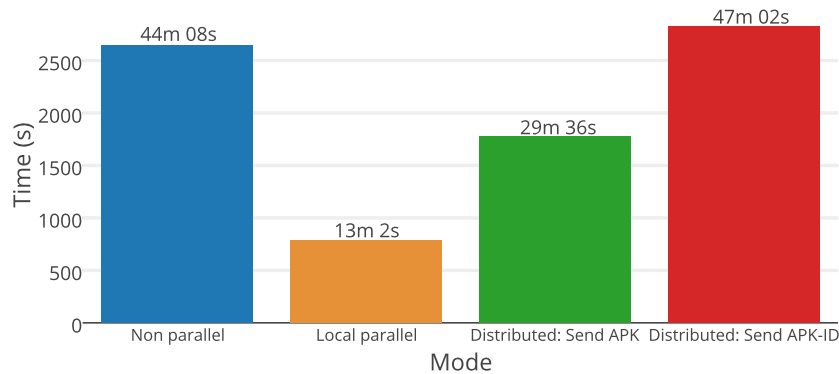


Figure 5.12: APK distribution

The result synchronization from *MongoDB* to the initiator of the analysis has been disabled in this experiment because it puts more load on the database and may influence its APK distribution. The results shown in Figure 5.12 indicate that the local parallel mode is faster than the distributed parallel mode, no matter which APK distribution strategy is used. Integrating the APK into the message that is sent to *RabbitMQ* allows preloading the message, because every process reserves one task. Prior importing of the APKs into *MongoDB* and only sending the identifier of the application forces the worker to fetch the APK from *MongoDB* before it can start the analysis. Moreover, the APK has to be build together from all of its *chunks*⁵². The distribution via *MongoDB* is 17 minutes and 26 seconds slower than the distribution via *RabbitMQ*.

Nevertheless, the *distributed parallel* mode is valuable, since *MongoDB*'s *sharding* facilities can be used to distribute data records among several instances. Having enough *MongoDB* nodes can circumvent the bottleneck that can occur while serializing and sending apps from the analysis initiator over the network. Furthermore, the caching behavior of *MongoDB* allows *MongoDB* to be used as an in-memory database. In the experiment, *MongoDB* was stopped, all unused pages removed and then the service was started again to prevent this effect.

Overall Performance

Now follows a presentation of an overall performance evaluation of *AndroLyze*. Figure 5.13 shows the results of executing a single script on the data set *Top Free 500 Archive* on 7 computers, each running 8 *Celery* processes.

The first script (left) does not need any analysis objects. Thus, job execution is much faster than loading the APK data from the hard disk drive and transferring it over the network. Obviously, increased script requirements increase the runtime. *ClassListing* (i.e., list all classes) and *ClassDetails* (i.e., list all classes and additionally show fields and methods) do not need any cross-references because they only access the disassembly. Creating cross-references, as needed by *CodePermissions*, *WebView* (i.e., list all methods

⁵²*AndroLyze* sets the chunk size as large as possible, preventing more chunks than necessary

5.3 AndroLyze: Static Mobile App Analysis

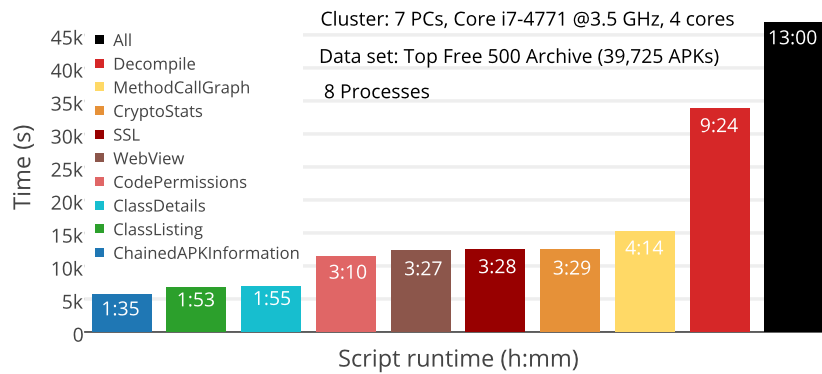


Figure 5.13: Performance of AndroLyze

using the `WebView` class), `SSL`, `CryptoStats` (see Section 5.3.5) and `MethodCallGraph`, adds more than one hour to the analysis time. Most of the time is needed to decompile all apps with *Androguard's DAD decompiler*, which consumes 9 hours and 24 minutes. Since there is an overhead for opening the APKs and providing the analysis objects, an analysis with all scripts where the objects are only provisioned once and shared, is 19 hours and 35 minutes faster compared to running all scripts alone.

AndroLyze@Work: Use of Cryptographic Code in Apps

To demonstrate the benefits of *AndroLyze* in a concrete analysis scenario, a study on the use of cryptographic code in APKs in the data set *Top Free 500 Archive* was conducted. The script checks for explicit calls to the block cipher modes ECB (Electronic Codebook) and CBC (Cipher Block Chaining) as well as for the most preferred ciphers, which in total took 3:29 hours to complete on 7 computers. The results are shown in Figure 5.14. They are in line with the results by Egele et al. [187], since many apps use cryptographic functions (65.65 %) somewhere in their code. Since there were snapshots of the Top Free 500 apps of all categories from different years, it was also analyzed which apps changed their cryptographic behavior, given they were still in Google's Top Free 500 apps. Throughout the years, AES (Advanced Encryption Standard) has always been the most popular cipher, and the study shows that most developers still favor AES. Overall, many developers requested CBC mode (87.51 %) independent of the used cipher, and only a fraction asked explicitly for ECB mode (still 20.35 %). Some apps use both modes in different areas of code, either for backward compability, as a fallback, for file format compability or for no apparent reason. For apps relying on ECB mode, the results show that 134 apps removed the ECB mode in their latest version. It is evident that checks like these and the ones performed by Egele et al. [187] should not only be performed once, but should be incorporated in a unit test-like fashion into build or release cycles. Having a standardized infrastructure, script, APK and result management as offered by *AndroLyze* helps providing audit trails for app lifecycles and various checks developed in-house, for the research community or for commercial entities.

5.3.6 Conclusion

In this section, a distributed framework called *AndroLyze* to support researchers in the process of conducting mass-audits of Android apps was presented. It was shown that

5 Security Vulnerability Analysis of Mobile Apps

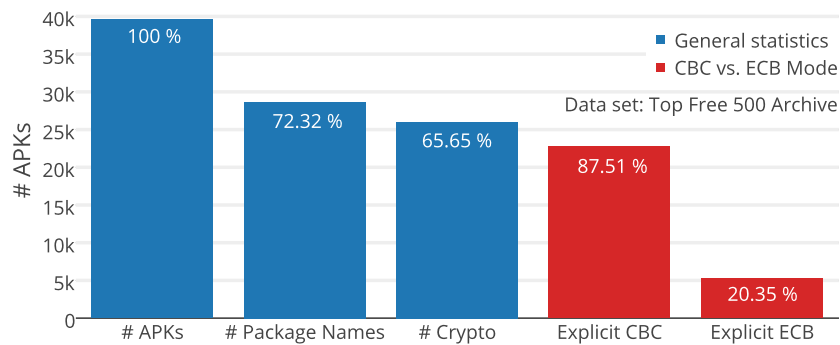


Figure 5.14: Crypto Statistics for Top Free 500 Archive

intelligent scheduling and APK management leads to faster results with potentially more useful information by incorporating past versions of apps. AndroLyze has been evaluated using the Top 500 Free apps from all categories of *Google Play* collected over three years, consisting of almost 40,000 apps requiring about 227 GB of storage space.

There are several areas of future work. Currently, the analysis scripts are based on *Androguard*, but AndroLyze can be used to easily plug-in other script backends for custom checks. Furthermore, the scheduling algorithms and script management facilities could also be used to optimize and streamline app runtime analysis. Finally, potential applications for *AndroLyze* include open repositories for static checks to bring the knowledge from the research community to app developers and into the development process of companies.

5.4 Dynalize: Dynamic Mobile App Analysis

5.4.1 Introduction

In the previous section, a framework for large scale static analysis of Android apps was presented. Unfortunately, not all app functionality can be identified in this way. Thus, certain bugs or vulnerabilities might be missed if relying solely on static analysis. This can be overcome by using dynamic analysis to observe runtime properties of a mobile app that is executed on a physical or virtual device. This process is very resource intensive, and it proves to be even more challenging to do automated tests for larger quantities of apps.

There are several tools that examine the runtime properties of mobile apps in order to evaluate bugs [194], performance issues [195] or to detect malware [196]. Furthermore, there are generic frameworks such as PUMA [197] that provide support for dynamic app analysis, but not with respect to scalability and dynamic resource provisioning. On the other hand, Device-as-a-Service (DaaS) providers such as Genymotion [198], ManyMo [199] and Testdroid [200] assume that a human tester controls devices *remotely*. These approaches are aimed at supporting the development and evaluation of *single apps* on a *small number* of virtual or physical devices in parallel.

In this section, *Dynalize*, a novel Platform-as-a-Service (PaaS) cloud for dynamic analysis of mobile apps, is proposed. Dynalize provides a scalable platform for dynamic app analysis, allows developers to integrate existing tools with only slight modifications and can be used to publish analysis results as Software-as-a-Service (SaaS) offerings. Therefore, Dynalize provides (i) a container management/job processing engine that serves as a layer between Infrastructure-as-a-Service (IaaS) and PaaS, and (ii) a service platform for the dynamic analysis of mobile apps.

Dynalize has the following features:

1. It runs virtual devices on top of IaaS instances, which provides scalability and dynamic resource provisioning.
2. It makes use of containerization and container virtualization, which allows developers to execute existing tools with heterogeneous dependencies regarding libraries and programming languages.
3. It provides a storage architecture for mobile apps, which can be used to distribute and process large amounts of data for thousands of apps in an efficient and coordinated manner.
4. It offers users (i) one-time and (ii) long-term provisioning, which enables users to either (i) start and terminate IaaS instances on demand or (ii) keep IaaS instances running in order to make use of local caches.
5. It offers a web service frontend to users, which allows users to publish their results as a SaaS service.

Parts of this section have been published in [11].

5.4.2 Related Work

Dynamic analysis has been used to analyze mobile apps with respect to privacy [201], security [196], [202], performance [195], energy consumption [203]–[205] and bugs [194], [206]–[209]. Several frameworks for large-scale dynamic analysis for mobile apps [197],

[210] and automatic test frameworks running in the cloud [211], [212] have emerged. Furthermore, on-demand physical and virtual devices for testing individual apps with different operating system releases and configurations [198]–[200] and clouds of devices for automatic tests [213]–[215] have been developed.

The problem of performing dynamic analysis of mobile apps has been solved differently in the past. Approaches such as AppDoctor[207] and AndroidRipper[206] analyze mobile apps sequentially and accelerate virtual device startups with snapshots. While this is practicable for a small number of apps, without parallelization, dynamic analysis on a large-scale can take months or even years. For example, a measurement study of the Google Play store by Viennot et al. [216] has covered roughly 1 million apps. Assuming an execution time of one minute per app, a sequential analysis of these apps would take about two years.

Another approach is to perform dynamic analysis in parallel on a cluster of physical [197] or virtual devices [210]. For example, Andlantis[210] can analyze 3,000 Android apps per hour for malware on a cluster of 200 servers. Nevertheless, it relies on dedicated resources and only supports malware analysis. In contrast, Dynalize provides a generic solution with dynamically provisioned resources, enabling the user to analyze mobile apps with respect to other targets than malware, such as performance, bugs or energy consumption.

Furthermore, cloud-based approaches have been proposed for large-scale dynamic analysis. For example, Mahmood et al. [211] have deployed the Android emulator on IaaS instances to perform a large number of analyses in parallel. Furthermore, Ravindranath et al. [194] have developed VanarSena, allowing app developers to upload a Windows Phone binary and obtain a bug report within a short amount of time. In contrast, Dynalize supports multiple platforms and custom emulators such as the open source QEMU extension PANDA [217] that adds the ability to record and replay executions. More importantly, Dynalize provides caching and scheduling methods to enable the user to consider costs/performance trade-offs and to decide between one-time or long-term analyses. Furthermore, in contrast to previous cloud-based solutions, a layered file system has fast deployment times even if the program that performs the dynamic analysis changes or makes use of other libraries.

More recently, container virtualization has attracted the interest of researchers (e.g., REMnux⁵³). Containers can be deployed faster, compared to IaaS instances, and can be arbitrarily assembled from multiple storage layers. Container virtualization can also be beneficial for dynamic analysis: (i) Different virtual device emulators, programming languages and libraries can be easily composed to an execution environment for an analysis. (ii) Virtual devices can be recreated fast in order to provide a clean execution environment for an analyzed app. Therefore, Dynalize adapts container virtualization for its platform solution.

Several cloud providers already offer container virtualization services based on the docker container engine⁵⁴, such as the Amazon EC2 Container Service (ECS)⁵⁵ and the Google Container Engine⁵⁶. A user can acquire a cluster of containers that can be composed arbitrarily out of existing libraries, execution environments and – of

⁵³<https://remnux.org/docs/containers/malware-analysis/>

⁵⁴<https://www.docker.com/>

⁵⁵<https://aws.amazon.com/ecs>

⁵⁶<https://cloud.google.com/container-engine>

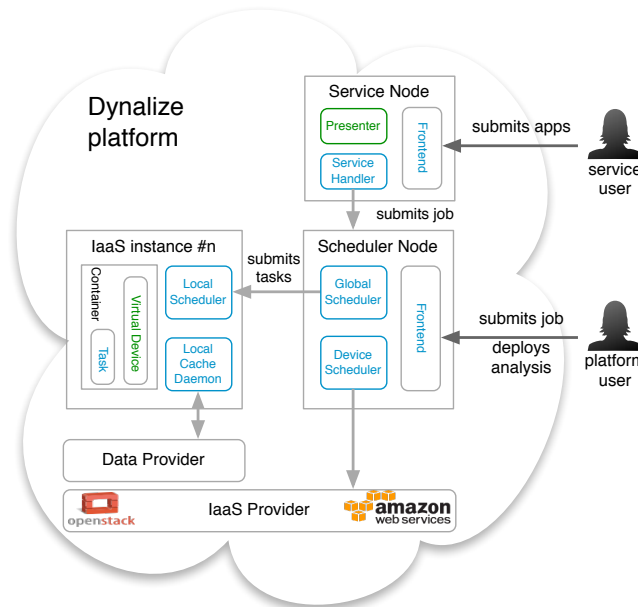


Figure 5.15: Dynalize platform architecture.

course – virtual devices. For example, a developer performing dynamic app analysis can start a cluster of IaaS instances and make use of a container cluster manager such as Kubernetes⁵⁷ to parallelize her dynamic analysis. But from a technical perspective, this approach is limited. Since these approaches leave the virtual server hosting the virtual container unmodified, the local storage cannot be used as additional cache, which is not efficient if large numbers of applications should be analyzed. For example, the study of Viennot et al. [216] is based on a data volume of 5.3 TB. Even smaller data sets with a volume of a few GBs require that the data is distributed and processed in an efficient and coordinated manner. Therefore, in contrast to pure container services, Dynalize provides a novel storage solution: It employs app prefetching and caching on the IaaS instance executing the Dynalize container, using a local cache daemon.

5.4.3 Dynalize's Design and Implementation

Dynalize consists of dynamically provisioned *IaaS instances*, a central *Scheduler Node* and a *Service Node* (see Figure 5.15). The components of Dynalize are described below.

IaaS Instances

Figure 5.16 shows the layout of an IaaS instance provisioned by Dynalize to analyze mobile apps. On the left side, the relation between data provider, local cache daemon and the container is shown. On the right side, the relation between global scheduler, local scheduler and local cache daemon is illustrated. Basically, the global scheduler partitions the total set of apps and sends a list of tasks to the local schedulers residing on the IaaS instances (see Section 5.4.3). An IaaS instance usually hosts two or more

⁵⁷<https://github.com/GoogleCloudPlatform/kubernetes>

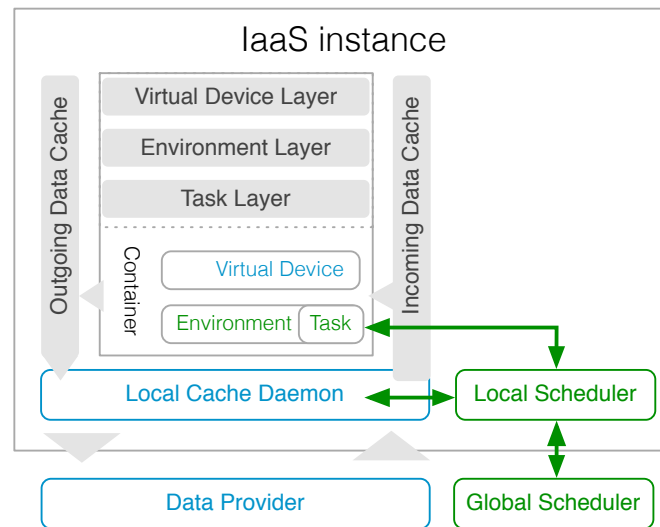


Figure 5.16: Layout of an IaaS instance used to analyze apps with Dynalize.

containers that are created and coordinated by the local scheduler. When a list of tasks is received, the local scheduler triggers the local cache daemon to asynchronously download these apps from the data provider. In the meantime, the containers are created and the virtual devices are started.

Container Layout In general, container virtualization makes use of operating system level virtualization: The host kernel is shared with several isolated process groups (in the remainder of the work referred to as containers or guests) that appear as separate physical systems to guest processes. In contrast to full- and para-virtualization solutions, container virtualization is lightweight; it enables isolated environments for user processes without additional virtualization overhead. The file system within a container is composed of several layers during creation time: Each layer represents a separate folder on the host's file system. During startup, these folders are overlaid transparently and form a single coherent file system using a storage backend.

During the development of Dynalize, three different storage backends were evaluated: Advanced Union File System (AUFS)⁵⁸, LVM2 DevMapper⁵⁹ and VFS⁶⁰. AUFS is a union file system that provides a file-level Copy-On-Write (COW) mechanism, while DevMapper is the kernel part of the LVM2 logical volume system, implementing a block-level COW. VFS does not provide COW support, instead a container's file system is formed by making a deep copy of its layers. The main advantages of composing a file system consisting of layers are deployment time and reusability of the base layer. Since AUFS gives the best results with respect to container startup times and throughput (see Section 5.4.4), Dynalize makes use of AUFS.

A Dynalize container consists of the following layers: (i) The virtual device layer, (ii) the environment layer and (iii) the task layer. The term *task* refers to the analysis of a single app, while the term *job* refers to a list of apps to be analyzed. The virtual device

⁵⁸<http://aufs.sourceforge.net/>

⁵⁹<http://sourceware.org/lvm2>

⁶⁰<https://www.docker.com/>

layer contains the emulation program for the virtual device and its virtual storage, i.e., its system image and a virtual SD card. This layer may contain emulators such as the Android Emulator, but also software such as the open source QEMU extension PANDA [217]. Dynalize virtual devices rely on full system emulation, since they are executed on the application layer, and hardware-acceleration is usually not supported in an IaaS cloud. On the other hand, full system emulation supports the execution of both native ARM or x86 binaries, which is not the case with hardware-accelerated x86-emulators. Furthermore, Dynalize offers different virtual device types, reflecting different amounts of resources reserved for each virtual device. The environment layer consists of programming languages, libraries and other dependencies for the execution environment of a dynamic analysis. It contains dependencies that are consistent for a long period and not changed with every minor update of an analysis. Finally, the task layer contains the program that performs the dynamic analysis.

This layered design ensures that Dynalize can use the snapshot capability provided by several IaaS providers. Even if a task changes slightly, only the task layer has to be retransmitted to the IaaS instances. Since the virtual device layer and the environment layer can grow very large (several GBs for system images, SD cards etc.), this container layout avoids the retransmission of large files within the virtual device and the environment layer. In contrast, if a new version of an analysis is deployed in Dynalize, it is retransmitted to and cached within each IaaS instance.

Local Cache Daemon In contrast to cluster solutions such as Andlantis [210], data transfer from and to the data provider is completely separated from the containers. By design, a data provider can be any storage service that supports downloads via HTTP/HTTPS, such as Amazon S3⁶¹ or GitHub⁶². All containers share a common read-only directory residing on the host, in which prefetched apps are stored by the local cache daemon. Prefetching allows a significant reduction of startup time of an analysis, since data transmission can be started before the container is created. Also, the incoming data cache allows the reuse of a previously fetched app. On the other hand, the outgoing data cache is a writable directory residing on the host, which is created during the container startup process. Each container has its own outgoing data cache directory, and when a task within a container signals the end of an analysis, the data within the cache is asynchronously transferred to the data provider while the container is restarted for the next analysis.

Scheduler Node

The Scheduler Node provides a frontend to the user, enable her to deploy her analysis and to submit jobs (see Figure 5.17). The user can choose between (i) one-time and (ii) long-term jobs, enabling her to either (i) start and terminate IaaS instances on demand or (ii) keep IaaS instances running to make use of local caches. Furthermore, a job configuration specifies a data provider as well as the type and number of virtual devices used to process this job.

⁶¹<http://aws.amazon.com/s3>

⁶²<https://github.com>

5 Security Vulnerability Analysis of Mobile Apps

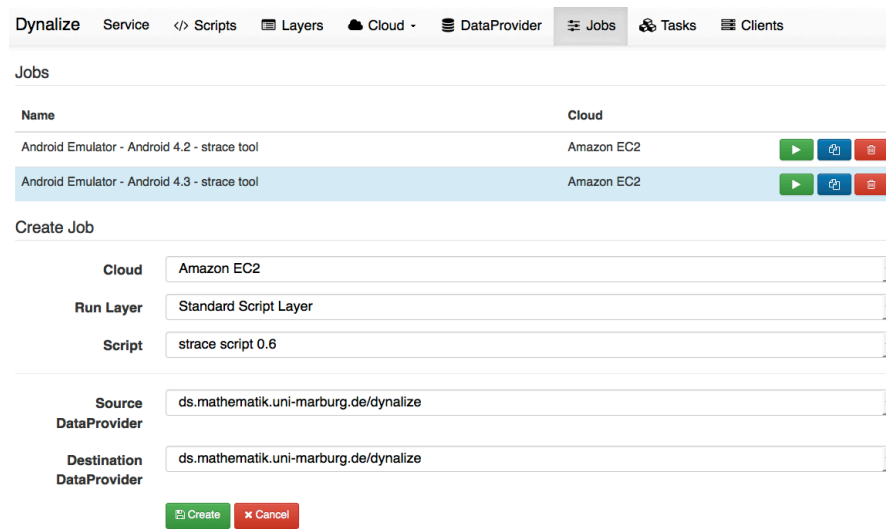


Figure 5.17: Screenshot of the Dynalize interface.

Device Scheduler IaaS instances are started and terminated by the device scheduler that maps virtual devices onto IaaS instances. It is based on a multi-objective scheduling algorithm: Since IaaS instances are billed in a pay-as-you-go manner, the objective function of the scheduling algorithm includes job execution time and execution costs. The search space includes different IaaS providers, each allowing their users to choose from a broad variety of instance types with different prices per hour. Primarily, instance types specify resources like the number of CPUs and the amount of RAM for a virtual server. Secondly, instance types also reflect different kinds of hardware, which are optimized, for example, for I/O-intensive or CPU-intensive workloads. Therefore, the multi-objective scheduler models execution time as a function of performance parameters like average virtual device startup time and throughput. The device scheduler is implemented as a variant of the Strength Pareto Evolutionary Algorithm (SPEA2) [218], has good convergence properties and is adaptable to different application areas.

Global Scheduler The global scheduler is responsible for distributing tasks to the IaaS instances. First, the tasks are partitioned among the virtual devices. The task list for each virtual device is sent to the corresponding local scheduler that initiates the app download and the creation of the containers (see Section 5.4.3). After a task is finished, the local scheduler notifies the global scheduler, which in turn keeps track of the on-going and finished tasks. In case of mixed instance types, the global scheduler reschedules tasks between the instances. In case of one-time jobs, instances can be terminated if no tasks can be rescheduled.

Service Node

As Figure 5.15 shows, both Scheduler Node and Service Node provide a frontend to the Dynalize user. While the Scheduler Node frontend is used to configure the IaaS provider interface, to submit jobs and to deploy analyses, the Service Node allows Dynalize users to make their work publicly available. For example, when an analysis has been successfully performed on a large number of apps, the developer can automatically

generate both a job submission template for the Service Handler and a web page template for the Presenter. Both templates can be modified to ensure a cost-efficient execution or a custom web page layout.

Since the main work is done by the Scheduler Node, a service request is processed in a straightforward manner. The Presenter runs a web server, on which a service user can upload apps to analyze. After an app is uploaded, the Service Handler submits a job to the Scheduler Node, which is terminated after all apps are analyzed. When the Scheduler Node signals job completion, results are uploaded to the cloud storage and made available to the user on the web page of the Service Node. The service can be evaluated more detailed with the demo service deployment.⁶³

5.4.4 Experimental Evaluation

In the following sections, a use case and a performance evaluation of Dynalize are described.

Use Case

As a use case for Dynalize, it was investigated which Android apps potentially use malware obfuscation techniques. Rastogi et al. [196] have shown that typical anti-malware solutions heavily depend on static signatures and therefore are ineffective against simple program transformations (transformation attacks). In particular, programs that make use of reflection (e.g., using the Java Reflection API) or code encryption can make parts of an app binary unavailable for static analysis. On the other hand, dynamic analysis can give hints on which apps use obfuscation techniques. For example, dynamic analysis can easily detect whether native libraries are linked within a binary or not. Afterwards, these apps can be further investigated with other tools.

For Dynalize, this is an interesting use case: (i) The execution of native binaries requires full system emulation, which Dynalize provides. (ii) The analysis needs to be performed on a representative set of applications, i.e., thousands of apps. In our use case, we have analyzed about 6,000 free Android apps from the Google Play store, with an average size of 17.7 MB per app and a total size of 10 GB, for signs of obfuscation. While a sequential dynamic analysis of these 6,000 apps took about 148 hours (nearly one week), Dynalize can offer a significant speedup due to parallelization. (iii) An appropriate de-obfuscation technique for arbitrary apps requires a variety of tasks, such as scanning the application for native library use, dynamic library linking, use of the Java Reflection API, system calls and anomalous network traffic. These can be implemented using different libraries within the execution environment of Dynalize. (iv) The use case can give evidence about the significance of dynamic mobile app analysis; to make appropriate assumptions about real-world mobile apps, a representative sample of apps needs to be analyzed.

The dynamic analysis in this use case has been implemented using Python 2.7, the Android Developer Tools, the Android Emulator, and the Android Debug Bridge. Most of the data has been collected with the Linux strace tool that traces system calls and signals. The Android Debug Bridge is used to install, start and trace an app on the

⁶³<http://ds.mathematik.uni-marburg.de/dynalize>

5 Security Vulnerability Analysis of Mobile Apps

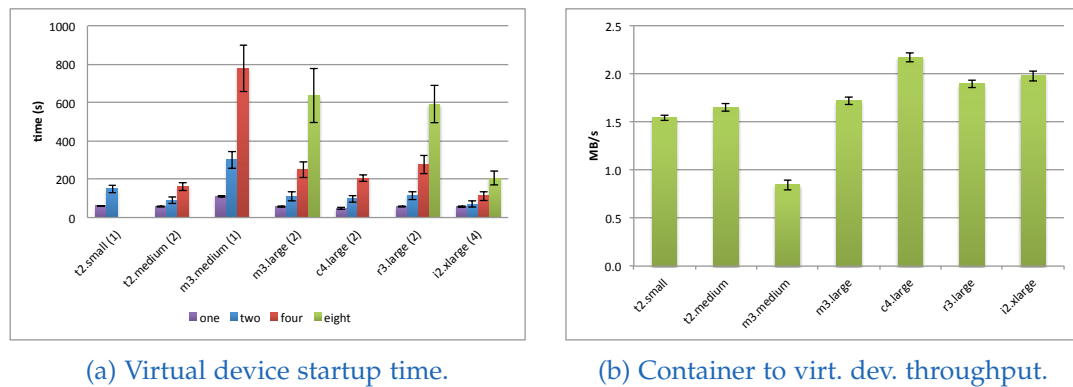


Figure 5.18: Virtual device startup time and container to virtual device throughput for different Amazon EC2 instance types.

virtual device. Each output is stored in a single file on the outgoing data cache on the IaaS instance.

Dynalize has been used with 40 virtual devices in parallel on 20 c4.large Amazon EC2 instances, two virtual devices each. The data was previously uploaded to Amazon S3. The additional startup time introduced by the instance boot time is negligible (less than a minute). However, due to resource sharing between virtual devices, the analysis of an individual app is approximately 10% slower than with a single virtual device. In summary, dynamic app analysis for the 6,000 apps was executed within 4.4 hours (3% of the time required for sequential analysis), with costs of 11.88 \$ (90 instance hours). The results show that 906 apps (15.1%) use native libraries and 192 apps (3%) dynamically load other binaries. There is a high usage of the Java reflection API, since it is used by 4,238 apps (71%). Another 3,483 apps (58%) make use of the Java cryptography API.

Virtual Device Performance

IaaS providers allow users to choose between several instance types. Scheduling virtual devices on IaaS instances requires us to consider different IaaS instance types. During the sequential execution of the use case, it was observed that the virtual device startup took 22-47% of the total execution time. Furthermore, the throughput between virtual device and container is important. Therefore, both virtual device startup time and throughput were evaluated for different IaaS instance types. They were measured with the Android Emulator with 512 MB RAM and the Android Debug Bridge, running on top of an Ubuntu 14.04 Linux distribution.

The virtual device startup time for different Amazon EC2 instance types in region eu-west-1 is shown in Figure 5.18a, while the costs for the used instance types is listed in Table 5.7. All virtual devices were started simultaneously, the values in brackets reflect the number of vCPUs. Note that t2.small only provides 1 vCPU and 2 GB RAM, while m3.medium provides 1 vCPU and 3.75 GB RAM. The t2 and m3 instance types are generic instances, while c4 is optimized for data processing, r3 for RAM accesses and i2 for storage. Furthermore, t2 instances are burstable: According to Amazon⁶⁴, CPU performance of t2 instances is based on CPU credits. Initially, a t2 instance earns credits for 30 minutes of 100% utilization, and every hour credits for

⁶⁴<http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/t2-instances.html>

Table 5.7: Prices of the used instance types

Instance Type	vCPUs	RAM (GB)	\$/h
t2.small	1	2	0.028
t2.medium	2	4	0.056
m3.medium	1	3.75	0.077
m3.large	2	7.5	0.154
c4.large	2	3.75	0.132
r3.large	2	15	0.195
i2.xlarge	4	30.5	0.938

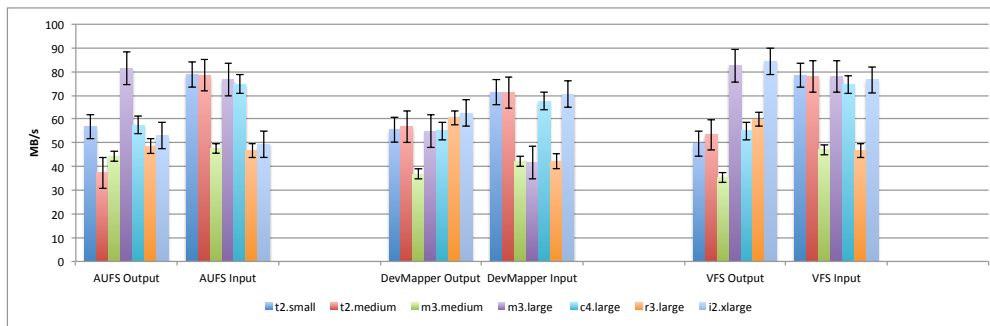


Figure 5.19: Container storage throughput for different EC2 instance types and storage backends.

another 12 minutes are added. If an instance runs out of credits, it is reduced to 20% (t2.small) and 40% (t2.medium) of CPU usage.

Since CPU performance is significant for full system emulation, Figure 5.18a shows the best results with 1+ vCPU per virtual device. Nevertheless, the values are not linearly scaled due to concurrent access on the device images (I/O). An interesting fact is that the best execution times were measured with both the low-end t2 and the high-end c4 instance types. This is due to t2's ability to burst CPU performance for a short period of time. Therefore, this is a good choice for analyses of a small number of apps. On the other hand, c4 instance types use a high-frequency Intel Xeon processor with SSD-backed instance storage. Therefore, both one and two virtual devices can be started within a short period of time.

Figure 5.18b depicts the throughput between the container and the virtual device for different Amazon EC2 instance types. It was measured by transferring a 133 MB app via the Android Debug Bridge from the analysis task to the emulated 2GB SD-card storage residing in the virtual device. The data is first passed through the ADB process running within the container and afterwards sent to a ADB server process within the virtual device via a QEMU-specific channel. Hence, the throughput of the Android Device Bridge makes heavy use of the CPU and involves several processes.

As Figure 5.18b shows, the best results were achieved with high-end c4 and i4 instance types. Due to the burst ability of t2, this low-end instance type also gives good results. The outlier for m3 indicates that the ADB throughput does not increase with a larger amount of RAM.

5 Security Vulnerability Analysis of Mobile Apps

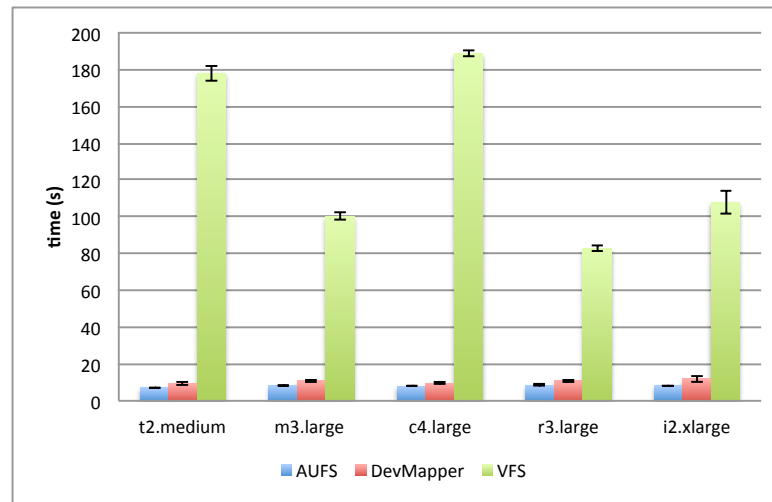


Figure 5.20: Container startup.

Container Storage Backends

As mentioned in Section 5.4.3, the container file system consists of multiple layers. This can be realized with three storage backends: AUFS, LVM2 DevMapper and VFS.

Figure 5.19 shows different backend throughputs measured with the bonnie++⁶⁵ file system benchmark suite. It tests sequential input and sequential output on the block level. Although the measurements with different Amazon EC2 instance types show a high variance, they indicate the following: (i) In general, read throughput per instance type is better than write throughput. (ii) AUFS is better for instance types with more CPU efficiency and more RAM, while DevMapper performs better for I/O optimized instance types. (iii) The best results are performed with VFS, which comes with nearly zero overhead.

Afterwards, the startup time for each container was evaluated. In this benchmark, a Linux ARM system image was booted with QEMU, measuring the interval between the container was created and the guest kernel was booted. Figure 5.20 shows that the AUFS storage backend performed best with an average startup time of 8.09 seconds. The DevMapper backend needed 10.55 seconds on the average, whereas VFS performed worst with an average of 131.68 seconds. The long startup time is due to the deep copy mechanism used by VFS: As Figure 5.21a depicts, Devmapper needed only slightly more (3.5%) disk space compared to AUFS, but VFS needed 1934% additional storage (32.12 GB in total).

The latter is also confirmed by Figure 5.21b that shows the average disk throughput of both sequential input and output. Since VFS cannot be used for its slow deployment times, an interesting result is that AUFS is slightly better than DevMapper.

Discussion

The results of the experiments can be summarized as follows: (i) In contrast to a sequential mobile app analysis, Dynalizer offers parallel dynamic mobile app analysis to process mobile apps on a large scale in short time. Furthermore, Dynalizer can be used

⁶⁵<http://www.coker.com.au/bonnie++>

5.4 Dynalize: Dynamic Mobile App Analysis

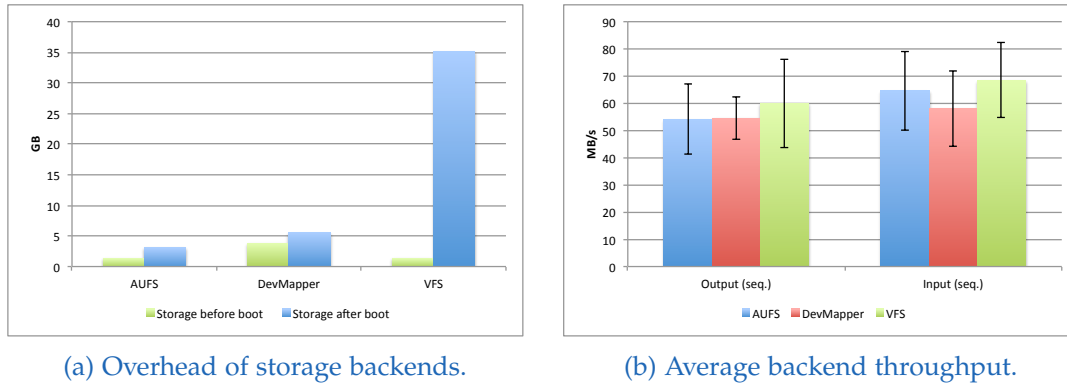


Figure 5.21: Average throughput and overhead of storage Backends.

as an economical and flexible alternative to a dedicated cluster. (ii) The measurements with different Amazon EC2 instance types indicate that the ability of $t2$ instance types to burst CPU performance for a short period of time is useful for analyses of a small number of apps, while high-end instance types like $c4$ can perform analysis efficiently with a high number of virtual devices in parallel. (iii) While throughput and virtual device startup measurements show high variance for different instance types, the AUFS storage backend proved to be the best of the three alternatives AUFS, DevMapper, and VFS.

5.4.5 Conclusion

In this section, Dynalize has been presented, a Platform-as-a-Service (PaaS) cloud for the dynamic analysis of mobile applications. It relies on container virtualization on top of IaaS instances, enabling dynamic provisioning and fast deployment of dynamic analyses. A platform architecture as well as a custom container layout and a novel storage solution on the virtual server layer have been presented. A dynamic security analysis of about 6,000 Android applications has shown the cost- and runtime-efficiency of a large-scale analysis for thousands of apps. Experiments focusing on container startup, virtual device to container throughput and different storage backends showed the feasibility of our approach.

Future work will address a deeper comparison between Dynalize and container virtualization engines like Kubernetes. It will also address other storage backends, like `btrfs`⁶⁶ and `overlayfs`⁶⁷. Furthermore, efforts will be made to improve the cost- and runtime-efficiency of Dynalize. For example, QEMU checkpoint/restore mechanisms will be evaluated as an alternative to full virtual device termination/restarts. Finally, it will be evaluated how Dynalize can be automatically customized to support new device emulators and the corresponding environment libraries.

⁶⁶https://btrfs.wiki.kernel.org/index.php/Main_Page

⁶⁷<https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/Documentation/filesystems/overlayfs.txt>

5.5 Security Assessment of Emergency Apps

5.5.1 Introduction

Since the usage of traditional broadcast media, such as television and radio, is declining, apps running on mobile devices play a major role in reaching the masses with alerts and warnings. Digital distribution of information has many benefits, but also presents new opportunities for malicious people. The propagation of fake news or false alarms can be used to trigger certain behavior, worsen an emergency situation, or cause an emergency situation in the first place. For example, a spoofed, fake alert can trigger a mass panic and disrupt the public order. How vulnerable the Internet infrastructure is can be seen by incidents like the temporary rerouting of Telegram traffic through Iran in 2018.⁶⁸ Sophisticated attackers such as federal authorities or governmental institutions easily have access to legitimate certificate authorities and can attack network links world wide. Therefore, the focus of this section is on a security assessment of common disaster warning apps and whether it is possible to spread false alarms, under which circumstances and for which attackers. Furthermore, apps have access to sensitive user information, including their location, which should also be protected. More troublesome, a warning app might just as easily be used to spy on citizens.

5.5.2 Popular Emergency Apps

The security audit focuses on official and popular emergency warning apps, partially consisting of national German and/or international apps, where the international apps either have a world-wide focus or focus just on the USA. A brief description of each of these apps is given below. All install base numbers are from July 2018.

German Official Apps

DWD WarnWetter The federal weather service (DWD) provides this app free of charge to issue weather forecasts and warnings to the general public. While the API for accessing the weather and warning data is made public, the app itself is closed source software. The install base on Android alone is over one million devices.

BBK NINA Similarly, The Federal Office of Civil Protection and Disaster Assistance (BBK) has an app to distribute public alerts, including weather related warnings. Again, the data itself is openly available, but not the source of the app. According to the Google Play Store it is deployed on over one million mobile devices.

KATWARN This app has a similar focus as NINA but lacks a public API. Due to the federal system in Germany, some cities, counties or states rely either on NINA or KATWARN. It is installed on over one million Android devices.

Sicher Reisen The German foreign ministry developed this app to inform and warn citizens abroad about various dangers and disaster scenarios. According to the developer, it has been installed on over 400.000 devices so far.

⁶⁸https://www.theregister.co.uk/2018/08/01/bgp_route_leak_telegram_iran/

BIWAPP The *Bürger Info & Warn App* is a free app developed by *Marktplatz.GmbH*. It can be used by cities, municipalities and professional responders to publish all kinds of information and warnings. Besides weather warnings or larger disasters, it can also publish topics including cancellation of school classes, accidents, or bomb alerts. Currently, it is installed on more than 100.000 Android devices.

International App Selection

FEMA The US Federal Emergency Management Agency also has an app to inform and warn the public about various dangers. Furthermore, it provides general advice and information about the location to nearby emergency shelters. So far, it has been installed on over half a million Android devices.

Disaster Alert The Pacific Disaster Center developed an app to inform about active hazards world wide. These hazards can be of natural origin, such as earthquakes, tsunamis or wildfires, or man-made disasters. The app was downloaded over half a million times from the Play Store.

NOAA Weather Radar This is a commercial offering, providing US and international weather forecast services including various warnings. According to the Google Play Store, this app was installed on over 10 million devices. This makes it the app with the largest user base in this selection.

5.5.3 Common Attack Surface

While the audit was performed with semi-automatic scripts, utilizing static and dynamic analysis, for various common vulnerabilities, they were also manually inspected to eliminate false positives and to search for additional security bugs. The most common problem areas that were looked at are briefly described in the following.

Transport Layer Security As already shown in Section 5.2, many mistakes can be made regarding transport layer security in mobile apps. While spreading public information is not reliant on encrypted links, as the content is public anyhow, it still helps to protect the content from being tempered with while in transport. All apps rely on HTTP and therefore should use TLS to encrypt their links. Common mistakes here include:

- Accepting self-signed or invalid certificates
- Accepting any valid certificate without hostname verification
- Not pinning the expected certificate

Without implementing certificate pinning, a user is still vulnerable to any sophisticated attacker that can manage to install a malicious root certificate on a user's device or has access to any already trusted certificate authority, which can easily be achieved by governments in a cyber-war or long planned cyber-terrorist attacks.

5 Security Vulnerability Analysis of Mobile Apps

Integrity of Messages After successful delivery of alerts, there is still the possibility that someone unauthorized placed the false information on the server or somehow managed to change the message in transit. Therefore, message integrity is not only relevant for transmission, but also for further processing on the device. For this evaluation, it must be checked whether the published data is digitally signed, where the certificate comes from, and whether the client verifies that the message comes from a legitimate source.

App vs. App - Intent Security Besides altering information on the server or while it is transmitted over the network, modern mobile device operating systems such as Android provide various ways of Inter-Process Communication (IPC). Using intents and receivers, an app can expose its functionality to other local apps on the device. This additional attack surface might be used by a malicious app to trigger custom alerts in a vulnerable app or impersonate the legitimate app.

Privacy Leaks All of the mentioned apps have access to potentially sensitive information, such as camera images, GPS locations, contacts, or device resource usage. Protecting this data should always be a high priority. Therefore, we investigate whether this data is used locally or transmitted to a remote destination and under which circumstances this happens, e.g., due to crash reports, regular checks for warnings etc.

5.5.4 Individual App Audits

All apps were audited with the above four topics in mind. Therefore, a short summary matrix for each app is provided, displaying the state of *Transport Layer Security* (🔒 TLS usage, 📌 certificate pinning), *Message Integrity*, *IPC Security* and *Privacy* (🗑️ leaking data). For each category, an app can receive a score of up to 4 points if no issues were found and countermeasures have been implemented in all categories or no attack surface is given. This results in a maximum score of 16 points that an app can achieve. If an exploitable vulnerability is found, screenshots with a false alarm are given, including out-of-place strings such as "MITM" or the number "23" as an indicator for a successful attack.

DWD WarnWetter

Security Matrix: <i>DWD WarnWetter</i>			
Transport Layer Security:	🔒📌	Message Integrity:	❌
IPC Security:	✅	Privacy:	✅
Overall Score:	10		

The official German weather forecast and warning app fetches all its data via HTTPS. It was not possible to use a self-signed certificate, a broken one or a valid one for another hostname. But since no certificate pinning was used, any trusted authority can issue a valid certificate for a malicious server. Since the delivered data from DWD was not signed and therefore cannot be validated, a Man-in-the-Middle attack is still possible for a sophisticated attacker, as shown in Figure 5.22. Here, an attacker can not only inject text based warnings, but also directly embed videos, e.g., as a weather reporter presenting a false forecast or spreading a fake warning.

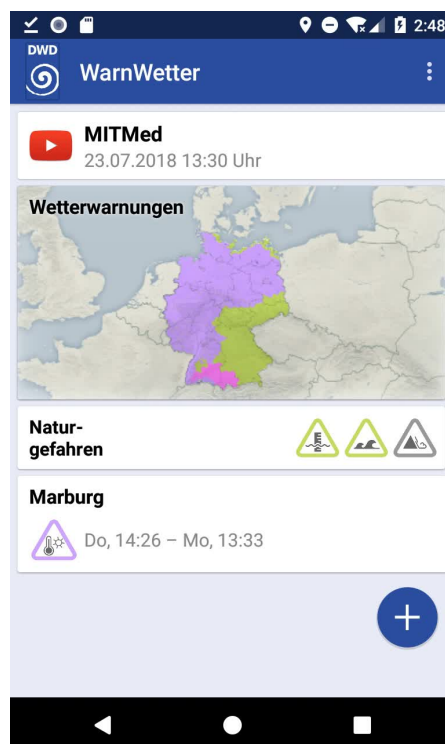


Figure 5.22: Successful MITM attack on WarnWetter.

Furthermore, the GPS location is used, but only locally, since all warnings are loaded from the server and filtered on the device for the ones relevant for the user. There is also usage and debug code present that gathers various statistics, but there seems no valid code path to trigger this behavior. Thus, no major privacy issues were found with this app.

BBK NINA

Security Matrix: <i>BBK NINA</i>			
Transport Layer Security:	🔒📍	Message Integrity:	✗
IPC Security:	✅	Privacy:	📞
Overall Score:	6		

Similarly to WarnWetter, NINA also relies on HTTPS to fetch its content. Trust managers and TLS code in general work as intended, but again, no certificate pinning and no signing of the retrieved data can be found. This enables advanced attackers to perform MITM attacks and inject their own alerts, as shown in Figure 5.23. Furthermore, hardcoded basic authentication credentials are stored in the app to communicate with one of the backend systems responsible for push notifications.

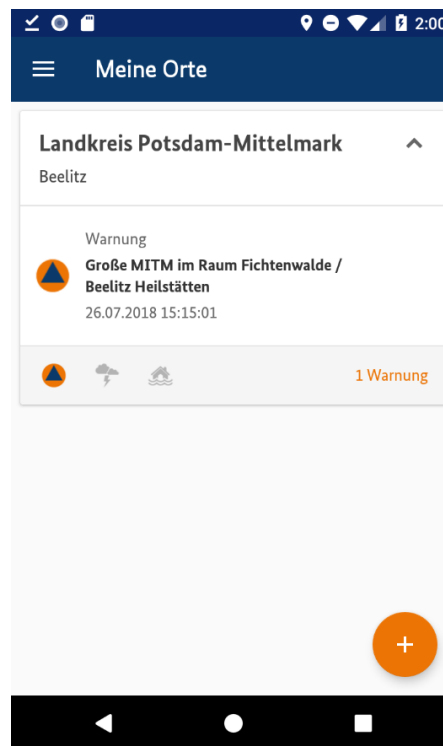






Figure 5.23: Successful MITM attack on NINA.

Privacy-wise, the whole warning database is downloaded and locally checked for relevant alerts. Yet, Google Analytics is contacted by the app, which should not be necessary for the functionality of the app. There is a lot of tracking code in place, giving detailed feedback about the device and app usage. It is possible to deactivate this code via the settings, by default it is turned on. From a privacy standpoint, this should be the other way around.






KATWARN

Security Matrix: KATWARN	
Transport Layer Security: 	Message Integrity: 
IPC Security: 	Privacy: 
Overall Score:	8

KATWARN takes a completely different approach than the previous two applications. Here, the warning database is not publicly available. Each user/device has to register with the service and subscribe to different alert topics (e.g., Oktoberfest, Dippemess) and/or regional areas. The app itself ships a certificate to verify the identity of the server. This means that for any successful MITM attack to be carried out, the app itself has to be heavily modified. This server pinning is an effective security measurement against network based attacks, even though the system still lacks signed alert data. Relying on complex server-side software for data exchange, on the other hand, adds unnecessary complexity and, therefore, provides a larger attack surface to the overall system.

Having each device uniquely register at the central server and requesting specific locations regularly means that identifying individuals and tracking them is rather easy. Although this is happening over an encrypted link, it is not necessary for the application to provide its functionality, as the previous apps have shown.

Sicher Reisen

Security Matrix: <i>Sicher Reisen</i>			
Transport Layer Security:			Message Integrity: 
IPC Security:			Privacy: 
Overall Score:	10		

Attacking the app on the transport layer works in the same way as it does with WarnWetter and NINA. No certificate pinning is in place and the data is directly rendered to the user, without any validation. It is possible to display any HTML text and external images, but not the execution of JavaScript. A successful attack is shown in Figure 5.24. Furthermore, hardcoded credentials together with basic authentication are used for each request to the server.

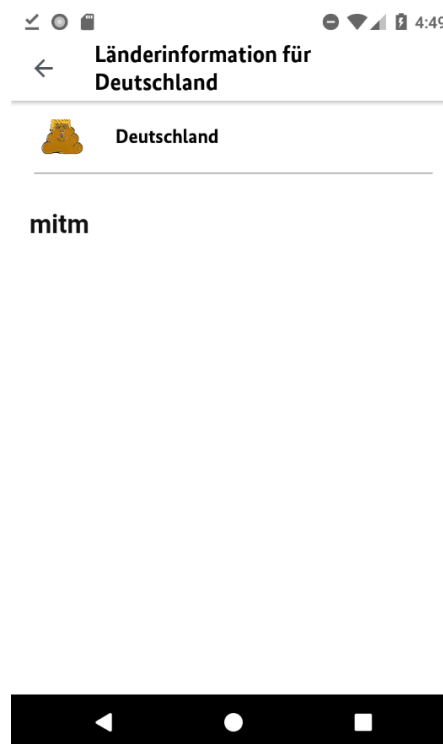


Figure 5.24: Successful MITM attack on Sicher Reisen.

No external resources are contacted besides those relevant for the main service. All issued warnings are downloaded and the relevant ones for the user's location are locally selected, as it should be. Overall, the app leaves a good impression in terms of privacy.

BIWAPP

Security Matrix: <i>BIWAPP</i>			
Transport Layer Security:	🔒🔴	Message Integrity:	❌
IPC Security:	✅	Privacy:	(✅)
Overall Score:	8		

The app also uses TLS correctly, but lacks certificate pinning and signing of the received data. Therefore, it can easily be tricked into displaying false information when a trusted certificate authority is used for the attack (Fig. 5.25). A few informational pages are loaded via HTTP to display information about pages and the likes.

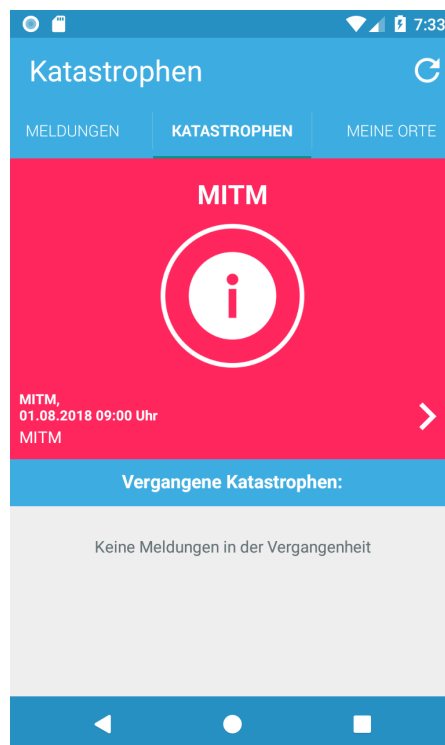


Figure 5.25: Successful MITM attack on BIWAPP.

While analyzing the HTTPS-based protocols, more possible attacks on the app and the server infrastructure came to light. Since each user is identified by an easily guessable 6-digit number, an attacker can trigger test alarms for any or all users by brute-forcing the numbers and sending a short POST request to the API server. The app downloads the complete alert database, with a current size about 1 MB, every time the app is started or the main screen is refreshed. Clicking on the test notification also triggers this behavior. Depending on the state of the app, the whole database is then downloaded multiple times for one test alarm. This can be used to deplete the bandwidth volume of the users phone contract or put stress on the server, resulting possibly in a denial-of-service attack. Furthermore, these brute-forced user IDs can be used to get the subscription list of any BIWAPP user, also revealing their current location if they did enable the

5 Security Vulnerability Analysis of Mobile Apps

guardian feature. During analysis of the requests and corresponding results, it became evident that some fail regularly due to syntax errors, producing stack traces in the server responses. These might contain useful information for attackers going after the server systems.

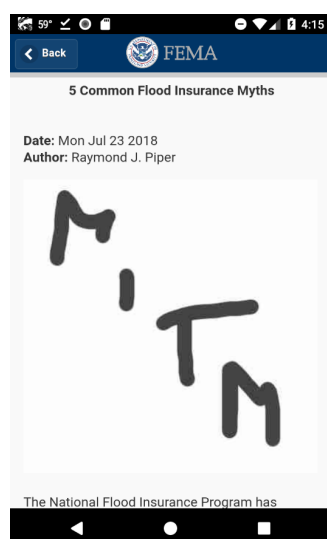
The app itself logs verbosely to the device system log but does not transmit this data back home. Privacy only leaks due to the bad protocol decisions and the server back-end security, where an attacker could gain the users location and subscribed interests. Furthermore, one of the external *about* pages uses piwik⁶⁹ to gather statistics, such as the user agent. This is only triggered if one explicitly clicks on the link in the *about* section. Thus, for BIWAPP the privacy issues are mostly related to the server component and bad protocol/API design, and not to the app code itself.

⁶⁹<https://github.com/piwik>

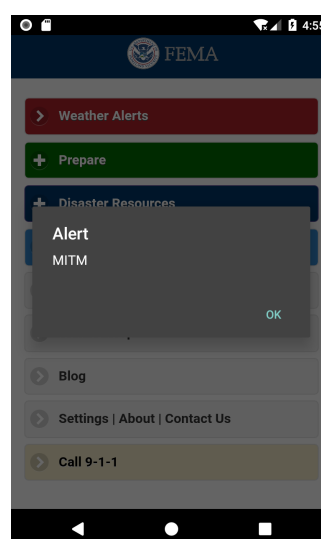
FEMA

Security Matrix: <i>FEMA</i>	
Transport Layer Security:	🔒📍
IPC Security:	✅
Message Integrity:	❌
Privacy:	📞
Overall Score:	4

The FEMA app partially uses TLS with correct hostname verification but also lacks certificate pinning. Therefore, it is possible to alter some content such as the blog news (Fig. 5.26a). Furthermore, some data is loaded via plain HTTP, such as the leaflet JavaScript library, which can easily be modified to display false data (Fig. 5.26b). This leaves the app wide open for easy manipulation, since not only text and images can be changed, but JavaScript code can be directly executed.



(a) Sophisticated attacker performing MITM attack.



(b) Primitive attacker changing map code during app start.

Figure 5.26: Different successful attacks on FEMA app.

Regarding privacy, the user's location and a possible target shelter is leaked via HTTP when activating this functionality. Also, a token is generated when launching the app for the first time. It is used for communication with the back-end server, but user- or device-specific data does not seem to be involved. Furthermore, the Android version and platform are leaked by HTTP(S) links.

Disaster Alert

Security Matrix: <i>Disaster Alert</i>			
Transport Layer Security:	🔒🔴	Message Integrity:	✗
IPC Security:	(✗)	Privacy:	🔴
Overall Score:	2		

Contrary to most of the previously mentioned applications that were vulnerable to MITM attacks only with valid certificates, PDC's Disaster Alert even loads HTML from unencrypted HTTP links. Besides static content, it is also possible to execute JavaScript code within the app. This makes injecting manipulated content very easy, even for unsophisticated attackers, as displayed in Figure 5.27. Furthermore, hardcoded credentials are used for basic authentication in some API calls to the service. Most of these calls are sent twice, once without credentials, thus failing, and once with credentials. This behavior wastes precious resources in a disaster scenario. Moreover, the app also exposes complex IPC functionality locally that under specific circumstances might be used to an attacker's advantage.

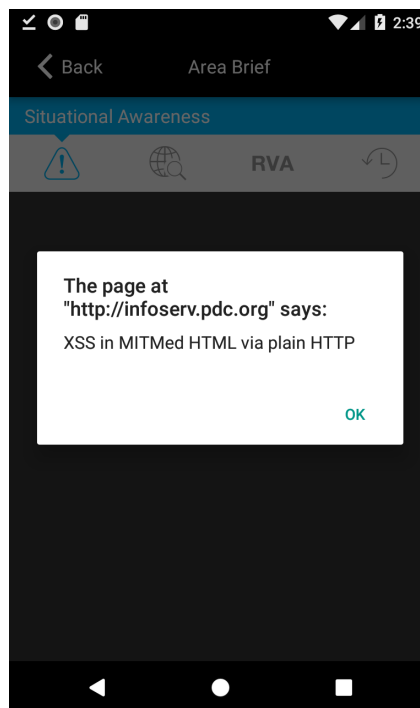


Figure 5.27: Successful MITM attack on Disaster Alert.

Privacy-wise, transmitting data unencrypted is also problematic since anyone with access to the routing path or local WiFi can easily eavesdrop on the user. These insecure transmissions happen for the main app code, but also for helper functions such as Google's geocode API. Google Analytics is used by the app when accessing the FAQ which also happens over an unencrypted link. Furthermore, privacy related information is sent to another analytics server via HTTP.

NOAA Weather Radar

Security Matrix: NOAA Weather Radar	
Transport Layer Security:	🔒📍
IPC Security:	✅
Message Integrity:	❌
Privacy:	📞
Overall Score:	4

This app uses plain HTTP to load an overlay for its weather radar on the map view. This can be used to replace the tiles with custom warning symbols, as shown in Figure 5.28. Weather warnings are also downloaded as a ZIP file over a plain text link. Manual inspection shown that the remote server is capable of communicating using HTTPS. Thus, it is really a bug in the app and could easily be fixed by simply switching the protocol in the source code. On the other hand, certificate pinning is used for some of the embedded functionality and specific servers, mostly integrated code from social and ad networks. Yet, the servers vital for the service itself are not pinned or do not use TLS at all.

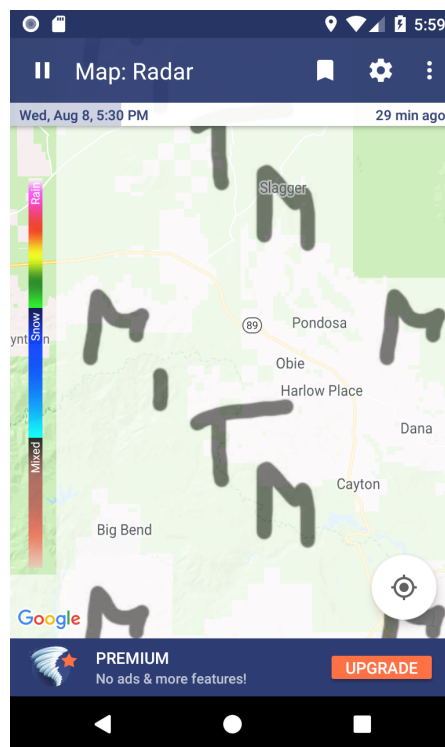


Figure 5.28: Successful MITM attack on NOAA Weather Radar.

The user's location is leaked via plain HTTP to the weather service provider. Also, the app contains code for various ad networks, potentially leaking information and broadening the general attack surface.

5 Security Vulnerability Analysis of Mobile Apps

Table 5.8: Audit summary grouped by attacker skill and general privacy issues.

App	Primitive Attacker	Sophisticated Attacker	Privacy
<i>DWD WarnWetter</i>		✗	
<i>BBK NINA</i>		✗	✗
<i>KATWARN</i>		(✗)	✗
<i>Sicher Reisen</i>		✗	
<i>BIWAPP</i>		✗	(✗)
<i>FEMA</i>	✗	✗	✗
<i>Disaster Alert</i>	✗	✗	✗
<i>NOAA Weather Radar</i>	✗	✗	✗

5.5.5 Conclusion

The security audit of these common emergency warning apps has shown that there are several shortcomings regarding security as well as privacy. Here, an ✗ indicates that an app fails to defend itself against the corresponding attacker, either with primitive skills or sophisticated ones, or has privacy issues in general. While a primitive attacker might try to intercept traffic with a self-signed certificate or by spoofing an intent to trigger a false alarm, most apps defend very well against these types of attacks. Given that a highly sophisticated and dedicated attacker can gain access to a certificate authority or manage to install own certificates in users trust stores, many applications display vulnerabilities. Furthermore, even when the transport layer is properly secured using certificate pinning, the data itself could be altered by an sophisticated attacker. This can only be prevented by digitally signing the data and verifying it at the end user's mobile device. Fortunately, most apps avoid security problems through improper protection of local IPC code. Only more or less harmless code is left exposed with a few exceptions of apps that already have shown serious flaws for transport security. Regarding privacy, almost every app has issues where it leaks data to third parties, to the official app servers or to Google.

Having strong transport layer security without signed data, e.g., KATWARN, can also cause problems during a disaster. While the publicly available data from WarnWetter and NINA can easily be served from another server, if the system is under stress by users or a denial-of-service attack, the KATWARN service requires more server-side logic and proper certificates, matching the one distributed with the app. Having a more complex API and the process of retrieving data from the server can easily be used for server-side attacks by sophisticated attackers. Hence, this should be avoided if it is not absolutely necessary. Of course, for data duplication to work, both, WarnWetter and NINA, should provide signature files for verification of their distributed data, which they currently lack. The importance of proper protocol design and remote server security also becomes evident when looking at the flaws found in BIWAPP. These lead to serious privacy issues and enable attackers to easily force massive resource consumption on clients, servers and network links, with minimal effort on the attacker side.

Overall, one can conclude that the official German warning apps provide basic security features, but fail to deliver proper security when facing more serious threats.

5.5 Security Assessment of Emergency Apps

The recent events of increased cyberwarfare, manipulation of votes and cyber terrorism in general have shown that there are major players easily capable of conducting such attacks on a large scale. The international apps have similar problems, but additionally also fail at defending against primitive attacks in various circumstances.

All vulnerabilities found during this audit have been disclosed to the vendors prior to releasing this thesis.

5.6 Summary

An analysis of 13,500 popular free apps from Google's Play Market regarding the state of SSL security and protection from Man-in-the-Middle attacks was presented in Section 5.2. MalloDroid was introduced, a tool to detect potential vulnerability against MITM attacks. The analysis revealed that 1,074 (8.0 %) of the apps examined contain SSL/TLS code that is potentially vulnerable to MITM attacks. Various forms of SSL/TLS misuse were discovered during a further manual audit of 100 selected apps that allowed successful MITM attacks against 41 apps and gathered a large variety of sensitive data. Furthermore, an online survey was conducted to evaluate users' perceptions of certificate warnings and HTTPS visual security indicators in Android's browser, showing that half of the 754 participating users were not able to correctly judge whether their browser session was protected by SSL/TLS or not. The section is concluded by considering the implications of these findings and discussing several countermeasures with which these problems could be alleviated.

In Section 5.3, *AndroLyze*, a distributed framework with unified logging and reporting functionality to perform security checks on large numbers of applications in an efficient manner, was presented. *AndroLyze* provides optimized scheduling algorithms for distributing static code analysis tasks across several machines. Moreover, *AndroLyze* can handle several versions of a single mobile application to generate a security track record over many versions. To demonstrate the benefits of *AndroLyze*, the Top Free 500 Android applications of all categories in *Google Play* collected over three years were analyzed. The resulting data set consists of almost 40,000 mobile applications and requires about 227 GB of storage space.

A Platform-as-a-Service cloud for the dynamic analysis of mobile applications, called *Dynalize*, was presented in Section 5.4. It allows researchers and developers to investigate mobile applications at runtime in a virtual device cloud and to publish the performed analyses as web services. In contrast to existing approaches, it makes use of container virtualization on top of Infrastructure-as-a-Service instances, enabling dynamic provisioning and fast deployment of dynamic analyses. A custom container layout and a novel storage solution on the virtual server layer ensures cost- and runtime-efficient large-scale analyses of thousands of apps. The applicability of *Dynalize* is demonstrated by a security analysis of about 6,000 Android applications. Experiments on container startup, virtual device to container throughput and different storage backends show the feasibility of the proposed approach.

The results of an security audit of the most common emergency apps was presented in Section 5.5. Here, it was shown that, while most apps provide basic security mechanisms, they still fall short when facing a sophisticated attacker. Furthermore, many apps also have privacy issues, leaking data to third parties or the app provider.

6 Secure Cloud Systems

6.1 Introduction

The previous chapter has shown that various security issues exist with existing mobile apps. This was also true for emergency specific apps. Furthermore, the audit of these apps has also revealed flawed protocol designs and bugs in the server APIs. To ensure the overall security, the backend systems running in the cloud must also be protected from attackers.

One of the main methods of communication for civilians as well as governmental and non-governmental organizations is still email, which also makes it a priority when rebuilding or installing emergency infrastructure.¹ Therefore, the security of the involved technologies such as SMTP are vital also during emergency scenarios. An in-depth analysis of the state of transport layer security of SMTP in the German IP-Space is given in Section 6.2.

Malware and kernel rootkits pose great threats to server infrastructure. The use of virtual machines especially in large hosting environments opens new possibilities for defense against these threats. In Section 6.3 a combined approach for live application tracing, signature based malware detection and kernel rootkit prevention is presented. This does not require any software installation in the virtual machine but only a modified kernel with the security critical code running "outside" the machine.

A holistic concept to detect, analyze and handle security anomalies in virtualized computing systems is presented in Section 6.4. The focus is on intrinsic security measures for virtualized cloud services. This is achieved by utilizing fast, minimal-intrusive sensors across all layers and using a federated Complex Event Processing (CEP) engine to aggregate and correlate events to find genuine attack and eliminate false positive alarms.

¹<https://www.heise.de/newsticker/meldung/Not-Internet-aus-dem-Ballon-3848035.html>

6.2 Assessment of Email Delivery Security

6.2.1 Introduction

Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL), are application-layer protocols to encrypt data segments transferred on the underlying transport layer of the Internet Protocol Suite. The communicating entities use X.509 certificates and thus rely on asymmetric cryptography to authenticate themselves and to exchange symmetric session keys to encrypt data flowing between the communicating entities. The use of X.509 certificates requires certificate authorities (CA) and a public key infrastructure (PKI) to verify the relation between a certificate and its owner, as well as to generate, sign, and administer the validity of certificates.

Several versions of TLS and SSL protocols are used in applications such as the WWW, electronic mail (email), and Voice-over-IP. The current version of TLS, TLS 1.2, was defined in RFC 5246 and released in August 2008, TLS 1.3 is currently available as a draft version. The most recent version of SSL, SSL 3.0, was released in 1996 (see RFC 6101²).

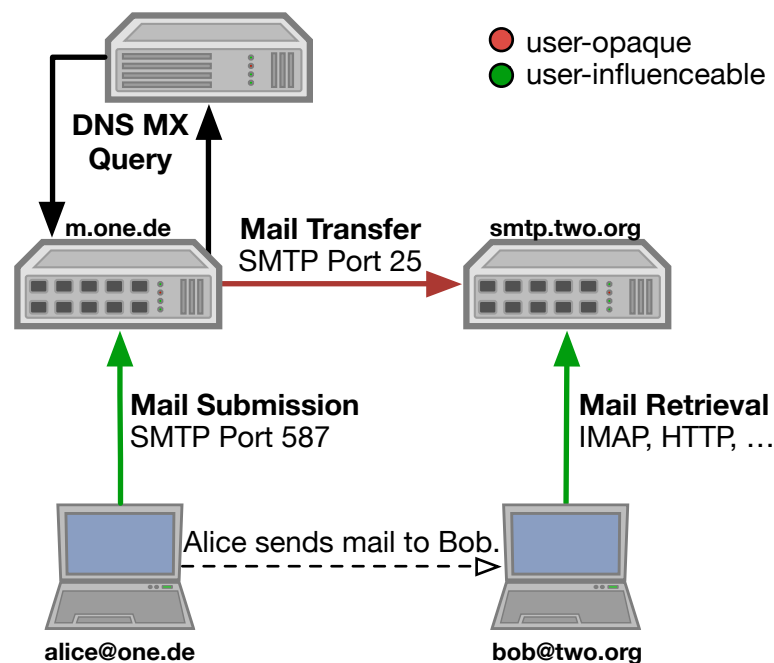


Figure 6.1: Email transfer and TLS usage.

Since email is a fundamental technology in everyday communication for government agencies, NGOs and civilians alike, its security is highly relevant during a disaster scenario.

The focus of this research is the use of TLS in SMTP^{3,4}, the Simple Mail Transfer Protocol, responsible for the delivery of email. In Figure 6.1, the process of sending and receiving email is outlined. Alice connects to her provider via SMTP on *Mail Submission*

²<https://tools.ietf.org/html/rfc6101>

³<https://tools.ietf.org/html/rfc821>

⁴<https://tools.ietf.org/html/rfc5321>

Port 587. Using *StartTLS*, she can encrypt the connection, as long as the email provider has this option enabled. After authenticating herself, she submits her email for Bob to her provider's email server. Her provider then looks up the DNS MX record for Bob's email address. In the next step, Alice's provider connects to Bob's provider using the *Mail Transfer Port* 25. Neither Alice nor Bob are able to review the connection properties the providers are using for the email transfer. Finally, Bob connects to his provider via the provider's web page or protocols such as POP3 or IMAP, and retrieves the email from his provider. Even if the email body may be encrypted by Alice using a client-side end-to-end encryption protocol such as Pretty Good Privacy (PGP) [219], meta-data such as sender, receiver and subject names may be visible to others, if the server-to-server connection is not encrypted properly. To secure the server-to-server connection, SMTP has been combined with TLS to encrypt email delivery and exchange between the participating entities⁵. Usually, the end user has no influence on this part except for his/her own mail submission to his/her provider's email server.

Recent revelations by Edward Snowden show that various government agencies actively and passively gather as much information from communication in the Internet as they can. Furthermore, since many corporate processes are coordinated using email within a company or with its costumers, the security of email is important for avoiding corporate espionage. Although consumers often communicate via Facebook, Whatsapp or Google Talk, email is typically used for banking, tax and online shopping related information that may be quite valuable for criminals, governments or other entities.

In this section, the results of a study of the security properties of SMTP over TLS conducted within the German IP address space (about 100 million IP addresses) is presented. A look at the involved cipher suites, the used certificates, CAs, and the general availability of TLS within the detected SMTP servers is taken. Since most private email correspondence is managed by a few big email providers, the behavior of their Mail Transfer Agents (MTAs) when communicating with improperly secured email servers is also analyzed. The results of the investigation lead to recommendations and best practices to solve some of the identified security issues.

Parts of this section have been published in [12].

6.2.2 Related Work

The security properties of the TLS/SSL landscape have been investigated in several works. The used certificates, the lengths of the private keys and the supported cryptographic functions bear significant security risks, as indicated by attacks such as POODLE⁶, BEAST⁷ and LUCKY THIRTEEN [220].

Lee et al. [221] have investigated cryptographic cipher suites, key lengths and support for the insecure version SSL 2.0 in TLS/SSL servers. Attacks on the RC4 stream cipher [222] and the MD5 hash function [223] have been presented in other publications. In their study on the certificate ecosystem used in the WWW, Eckersley and Burns [224] have shown that only around 40% of the investigated web servers had a valid certificate chain. In 2011, Holz et al. [225] have presented their analysis of the SSL landscape and the use of X.509 PKIs based on active and passive gathering of certificates, indicating that

⁵<https://tools.ietf.org/html/rfc3207>

⁶<https://poodlebleed.com/ssl-poodle.pdf>

⁷<http://www.hit.bme.hu/~buttyan/courses/EIT-SEC/abib/04-TLS/BEAST.pdf>

only 18% of the provided certificates were accepted without warning when validating them with the Mozilla Root Store. Ristic and Small [226] have presented an overview of SSL usage in the WWW. In 2013, a similar study has been published by Durumeric et al. [227] to analyze signing CAs, key lengths and cryptographic algorithms. In Section 5.2 we have conducted a mass audit of mobile Android applications to identify security issues in the use of TLS/SSL.

Giesen et al. [228] have published an approach to increase the security of recent mechanisms for TLS renegotiation. This hardening prevents Man-in-the-Middle attacks in some instances and minimizes the attack surface of applications using TLS. Focusing on TLS certificate management, Szalachowski et al. [229] have presented a solution based on the idea of publicly verifiable logs as made popular by Laurie et al.⁸ with Certificate Transparency for PKIs. The reference implementation for a HTTP environment aids in securing the PKI landscape as a whole. Ryan's work [230] also enhances Certificate Transparency and integrates it with end-to-end email encryption in an attempt to make it easier accessible for users and avoid some of the cumbersome quirks of PGP. This approach helps to improve certificate management, removes several trust issues and includes end-to-end encryption transport channel security. Even if certificate validation is performed accordingly, it still leaves an attack surface. A general overview of past attacks on SSL in the context of the WWW and issues with the certificate trust model has been shown by Clark and Oorshot [231]. However, the authors have web browsers and HTTPS traffic in mind, and their solutions are tied to this use case.

Huang et al. [232] have published a study to detect forged SSL certificates in the wild. They have analyzed over 3 million real-world SSL connections to Facebook. Even though the used detection mechanisms are limited, about 0.2% of the connections were detected using forged certificates. Validating certificates is an error-prone task, and various SSL libraries have different default behaviors. Automated tests have been performed by Brubaker et al. [233] to reveal major flaws in common libraries and how they are used in software like web browsers, giving false or at least misleading feedback to the user. Slow deployment rates of security fixes for SSL related code has been identified as a key problem by Bates et al. [234]. The proposed approach hooks SSL verification code to non-browser applications to give them increased security without tampering with the actual source code. These extra validations impose a 20 ms overhead and work out-of-the-box with 94% of Ubuntu's most popular packages.

None of the cited works is concerned with the use of TLS/SSL in SMTP server communications. However, email is still an integral part of today's business communications. Recently, Facebook⁹ has published some interesting observations regarding their email system. For example, *STARTTLS* is adopted by 76% of the unique MX hostnames that Facebook is in contact with. Moreover, 58% of the notification emails sent are successfully encrypted. The study concludes that general support for encryption is available, but proper certificates and certificate validation are missing.

⁸<https://tools.ietf.org/html/rfc6962>

⁹<https://www.facebook.com/notes/protect-the-graph/the-current-state-of-smtp-starttls-deployment/1453015901605223>

6.2.3 An Empirical Study of SMTP over TLS

The study of the email TLS landscape is based on the German IPv4 address space. In particular, 116,824,576 IP addresses were scanned to investigate the TLS properties of German SMTP servers. Using `nmap`¹⁰, it was first checked whether the relevant ports were open and then TLS versions, cipher suites, certificates, CAs, and email provider strategies were analyzed.

SMTP uses port 25 as its main port. Port 465 has been suggested for SMTP over TLS, called SMTPS (Simple Mail Transfer Protocol Secure)¹¹, but was later revoked; nevertheless some email providers still use this port. Port 587 is used to transfer email from the user to the provider's server.

Nmap also performs a reverse Domain Name System (DNS) lookup to find the domain names belonging to the referenced IP addresses. It is common to check the reverse DNS name when an email is sent to a server, thus the domain names should be set. These domain names are used later to check the validity of TLS certificates and to make sure that the IP in question is under the authority of the domain owner; emails from servers without valid reverse lookups should not be accepted.

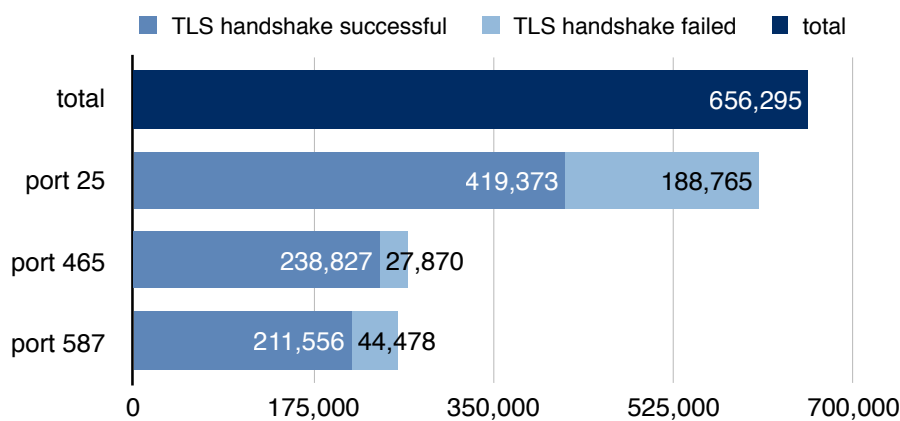


Figure 6.2: SMTP and TLS usage among the scanned hosts.

TLS Usage

From the scanned hosts reached at the 116,824,576 IP addresses, 656,295 hosts offer SMTP services on at least one of the following ports: 25, 465 or 587. Figure 6.2 shows that only 68.96% (419,373) of the 608,138 hosts that offer SMTP services on port 25 perform a successful TLS handshake on this port. On the mail submission port 587, 82.63% (211,556 of 256,034) complete a TLS handshake successfully. While on port 465, which has TLS enabled by default, a TLS handshake nevertheless fails in 10.45% of the attempts. It is likely that server administrators use this port for different services not related to SMTPS. The results presented below are either based on the 656,295 hosts that offer SMTP services on at least one of the three ports or on the 869,756 services (see Fig. 6.3) with a successful TLS handshake on at least one of three ports.

¹⁰<https://nmap.org>

¹¹<https://tools.ietf.org/html/rfc3207>

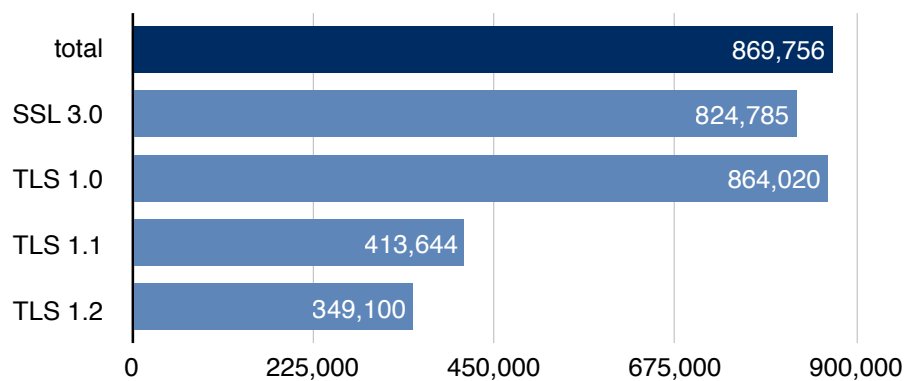


Figure 6.3: SSL and TLS versions used in the scanned services.

TLS/SSL Versions

TLS and SSL have a long history of organic growth, leading to a non-uniform use of their different versions. SSL 2.0 had certain design flaws, ultimately leading to an insecure protocol. SSL 3.0 was considered secure, until the POODLE attack went public in October 2014. The SMTP server scans took place in May 2014, five months before the POODLE attack was released to the public. On these grounds, the collected data allows us to observe the potential impact of the attack.

In Figure 6.3, the TLS/SSL versions used in the scanned SMTP servers are visualized. While the insecure version SSL 2.0 is not used at all, SSL 3.0 is enabled in 94.83% of the servers. The most popular TLS version 1.0 is supported by 99.34% of the servers, whereas the versions TLS 1.1 and TLS 1.2 are supported by less than half of the servers. BEAST's target was TLS 1.0, so there is no reason to hold back the newer versions. POODLE is also a good argument to use TLS 1.2 instead of SSL 3.0 – SSL 3.0 is only used for the sake (or curse) of compatibility.

TLS Cipher Suites

In a TLS handshake, the client offers a list of supported cipher suites to the server, which then picks one to secure the connection. To obtain all cipher suites supported by a server, the client has to iterate over the cipher suites and then tries to establish a connection using the selected cipher suite. This functionality is provided by the `nmap ssl-enum-cipher` script¹². Performing this test requires many connections to the tested servers and therefore creates quite some traffic in the network.

If an attacker is able to influence the TLS handshake by selecting a cipher suite that (s)he is able to break, this attack is called a cipher suite *downgrade attack*. Even if backward compatibility is a reason to offer potentially insecure cipher suites, they open up a huge attack vector. The cipher suite classification of the `nmap ssl-enum-ciphers` script provides three categories of cipher suites, as described below.

Broken Cipher Suites Broken cipher suites do not support protection against passive or active eavesdropping. Therefore, all anonymous cipher suites with an unauthorized Diffie-Hellman key exchange belong to this category. It is easy for an attacker to

¹²<https://nmap.org/nsedoc/scripts/ssl-enum-ciphers.html>

intercept and modify the messages between client and server and decipher the data. Furthermore, cipher suites using no encryption at all or just authentication are also considered as broken.

Weak Cipher Suites The category of weak cipher suites mainly consists of historical cipher suites. Examples are the *export ciphers* that emerged as a consequence of the US export rules on cryptographic systems before 1999, and cipher suites based on the legacy Data Encryption Standard (DES). Weak cipher suites can be deciphered in a brute force manner by powerful hardware in less than a day¹³.

Strong Cipher Suites All remaining cipher suites are strong cipher suites. Although there are known weaknesses in RC4 [235] and MD5 [236], they are nevertheless used with some workarounds in many cases. With RFC 7465, attempts are made to remove RC4 completely [237].

Another noteworthy property of cipher suites is Perfect Forward Secrecy [238]. It ensures that a session key derived from a set of long-term keys can not be compromised if one of the long-term keys is compromised in the future. Thus, an attacker cannot decipher past messages even with the server's private long-term key.

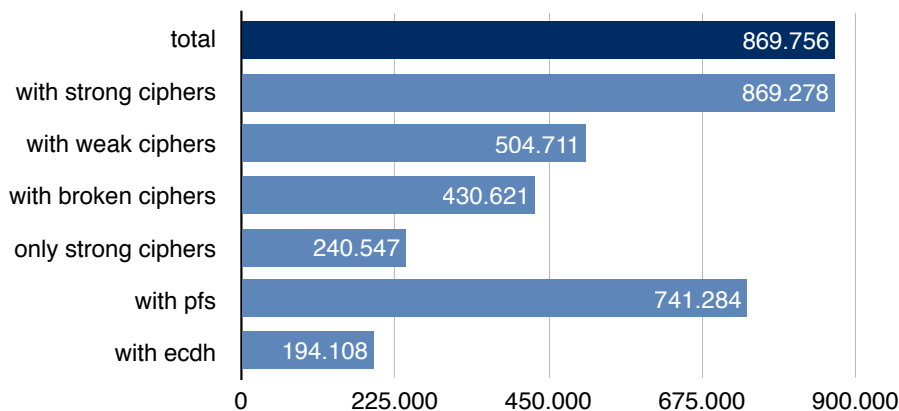


Figure 6.4: Use of cipher suites.

Figure 6.4 illustrates the used cipher suites. Almost every server offers strong cipher suites, but only 27.66% of the servers are hardened in the sense that they offer strong cipher suites only. It is remarkable that 85.28% of the servers offer Perfect Forward Secrecy (pfs). A widely discussed issue is the use of Elliptic Curve Cryptography (ECC), since many of the ECC algorithms are suspected to have been constructed under the influence of US intelligence services¹⁴. In our data set, 22,32% of the servers support ECC ciphers (ecdh).

In Table 6.1, the top ten most used weak and broken cipher suites are presented. Many servers accept weak cipher suites without the need to. Looking at the FREAK attack¹⁵ published in March 2015, more than half of the server's connections can be downgraded to weak or broken cipher suites.

¹³<http://www.sciengines.com/company/news-a-events/74-des-in-1-day.html>

¹⁴<https://www.wired.com/2007/11/securitymatters-1115/>

¹⁵<https://freakattack.com>

Table 6.1: Shares of weak and broken cipher suites.

Name	Usage	Share
DH_anon_AES_256_CBC_SHA	426,848	49.08%
DH_anon_3DES_EDE_CBC_SHA	426,835	49.08%
DH_anon_AES_128_CBC_SHA	426,779	49.07%
DH_anon_RC4_128_MD5	424,526	48.81%
RSA_DES_CBC_SHA	416,324	47.87%
RSA_EXPORT_RC4_40_MD5	410,627	47.21%
RSA_EXPORT_RC2_CBC_40_MD5	409,893	47.13%
RSA_EXPORT_DES40_CBC_SHA	407,480	46.85%
DHE_RSA_DES_CBC_SHA	382,903	44.02%
DHE_RSA_EXPORT_DES40_CBC_SHA	368,247	42.34%

TLS Certificates

The `nmap ssl-cert` script¹⁶ was used to retrieve the certificates used by the servers. This script tries to perform a TLS handshake with the given server/port and saves the Privacy-enhanced Electronic Mail (PEM) certificate in the scan report. In addition, it parses the fields of the certificate and stores them in the report. The script also supports the `STARTTLS` command for an active SMTP session.

The validity of a certificate is based on three major properties: issuance by a valid CA, time period of validity and a matching Common Name (CN). In addition, we investigate other issues of certificates, such as multiple uses of the key pair or short private keys. Lastly, all retrieved certificates are categorized using these properties.

Multiple Use of Key Pairs X.509 certificates are uniquely identified by their fingerprint using a MD5 or the *Secure Hash Algorithm 1* (SHA1) sum; current practice is to use a SHA256 hash nowadays. Although 656,295 hosts offer SMTP services in the data set, only 218,239 unique certificates were found. The reason is the presence of shared hosters using a single wildcard certificate for their hosts. This may not be a problem unless any user is able to retrieve the corresponding private key.

Another group of repeatedly used certificates are the certificates delivered with the servers' operating systems. Those are recognizable by their subject's CN, which often contains the operating system name, phrases like `localhost` or a domain ending in `*.local`. These configurations have to be considered insecure, because multiple users have access to the private key and therefore can decipher the TLS connections of other users.

Also private keys that are used multiple times in different certificates were found. At a first glance, this does not involve a decrease of security. However, if the private key is lost, one has to revoke not only one, but all certificates issued for this private key. There is no acceptable reason for a system administrator to use the same key for multiple certificates.

Time Period of Validity To investigate the time of validity, the certificates were categorized into four groups, as outlined in Table 6.2. A certificate has two timestamps,

¹⁶<https://nmap.org/nsedoc/scripts/ssl-cert.html>

Table 6.2: Time period of validity of the retrieved certificates.

Type	Count	Share
not yet valid	161	0.07 %
expired	56,078	25.68 %
valid	161,993	74.20 %
never valid	99	0.05 %

defining the time period during which the certificate is valid. 25.80% of the certificates were not valid. It is suspected these are hosts that are not maintained anymore. Although expired certificates may not be a security problem, they are an indicator that email is treated with a low priority. Expired certificates prevent secure communication with these servers if strict certificate validation is turned on at the email sender's side. There are 99 certificates which were never valid, since the *not-after* date of use was lower than or equal to the *not-before* date of use.

Table 6.3: Self-signed, server and CA certificates.

Type	Count	Share	is_ca	is_server
server	79,832	36.58 %	0	1
ca	705	0.32 %	1	0
self-signed	137,694	63.09 %	1	1
invalid	1	0.01 %	0	0

Certificate Issuance Also it was analyzed who issued and signed the obtained certificates. Table 6.3 shows that 63.09% of the certificates were self-signed certificates, i.e., certificates that are signed by the same entity whose identity they certify, by signing with their own private key. It was not looked into them in more detail, since the signature is not meaningful in these certificates, and they clearly represent a security issue.

To build the *chain of trust* for the non-self-signed certificates, the Ubuntu 14.04 trusted root certificates were used as a trust anchor. It were 24,641 certificates reachable by building a trust chain with the root and the retrieved intermediate certificates. This set of certificates corresponds to 11.29% of the 218,239 found certificates and can be considered trustworthy according to the signature.

Key Lengths Figure 6.5 shows the growth of key lengths in relation to the dates of issuance of their certificates. It is apparent that the key lengths are growing steadily, using an average of more than 2,230 bits at the time of the scans. The average lowest key length stems from 2007 with 1366 bits. It is remarkable that a 1023-bit RSA number has been factored in May 2007¹⁷, and that key lengths will become an issue when more powerful hardware becomes available.

¹⁷<http://phys.org/news98962171.html>

6 Secure Cloud Systems

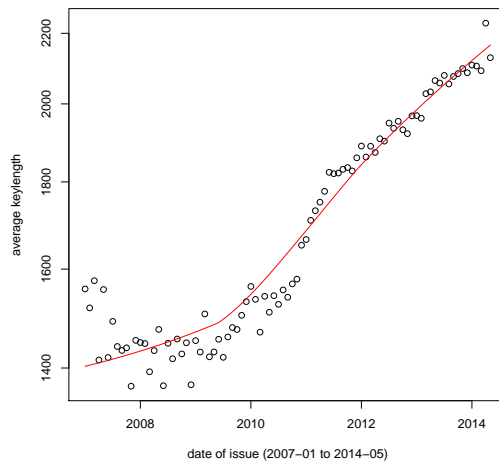


Figure 6.5: Change of the average key lengths over time.

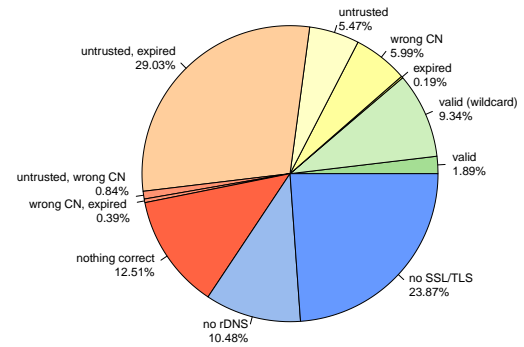


Figure 6.6: Categorization of services using the main security properties.

Categorization To summarize the findings regarding certificates, the three main validity properties to form eight disjoint categories of certificates are used. Since not every server has reverse DNS entries, it was not possible to get the names of 10.48% of the servers. 23.87% did not offer TLS. Figure 6.6 indicates that only 11.23% of all scanned email services had valid certificates in all concerns. Considering the certificate subject's CN, the category of valid certificates can be split up further. A certificate is not only accepted as valid, if the CN matches the domain name, but if it has a wildcard pattern in the form of **.domain.tld* matching the domain name (e.g., *mail.domain.tld*). Using this categorization, only 1.89% of the email services use a certificate only valid for this specific host.

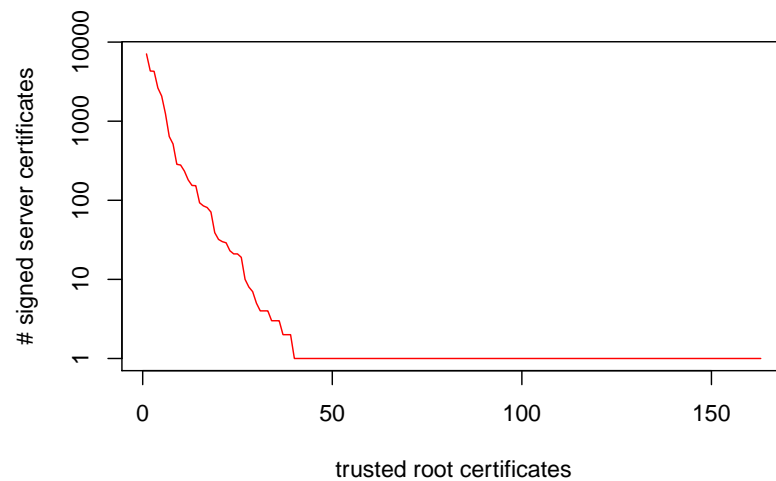


Figure 6.7: Root certificates in relation to their signed server certificates.

Certificate Authorities

A widely criticized problem of the TLS trust model are CAs. Over the last years, more and more CAs are trusted by software vendors. Figure 6.7 outlines the obtained trusted

Table 6.4: Top 10: Most popular CAs, measured by their issued certificates.

	Count	Share	Common Name
1	7,085	28.58 %	StartCom Certification Authority
2	4,309	17.38 %	AddTrust External CA Root
3	4,278	17.26 %	GeoTrust Global CA
4	2,632	10.62 %	thawte Primary Root CA
5	2,079	8.39 %	GlobalSign Root CA
6	1,243	5.01 %	DFN-Verein PCA Global - Go1
7	640	2.58 %	UTN-USERFirst-Hardware
8	515	2.08 %	COMODO Certification Authority
9	285	1.15 %	Go Daddy Root Certificate Authority
10	279	1.13 %	Deutsche Telekom Root CA 2

root CAs in relation to their signed server certificates of our test set. The ten most popular CAs shown in Table 6.4 sign 94.16% of the server certificates. To have 99% coverage, one needs to trust the 23 most popular CAs. The idea that "if a CA can sign for one domain, it can sign for any domain" leads to a loss of security. Examples such as the DigiNotar hack¹⁸ or the TurkTrust incident¹⁹ show that this is a problem of practical relevance. Since the default configuration remains unchanged in many settings, the operating system and application vendors should act more responsibly in this respect.

CA Topologies

To check the validity of a certificate, a user builds a so called chain of trust. In addition to its own certificate, the server can deliver additional intermediate certificates. The client then builds a chain of trust as follows: The issuer of a non-root certificate is identified by the issuer's properties of this certificate, and the signature is obtained by using the issuer's public key. To get a valid chain of trust, the last certificate needs to be in the list of trusted root certificates of the client. If no path from the server certificate to any of the trusted root certificates can be found, the certificate is considered as untrusted. This is often the case when a private CA for in-house deployment or self-signed certificates are used.

Using OpenSSL²⁰, a key-value database was built representing the graph of signatures and the hierarchy of CAs. Two CAs showed peculiarities. First, StartSSL uses two almost identical root certificates, only differing in the date of issuance and the serial number. The trusted root certificate set as shipped by default with Ubuntu 14.04 was used. Second, as shown in Figure 6.8, the Comodo CA uses more than just the CA and intermediate certificates. Comodo cross-certifies its root keys using the other root certificates. This results in multiple signed key pairs and therefore in multiple certificates for every key. If one of those root certificates is revoked, there are signing chains starting at other root certificates available. Comodo enables a kind of fail-safe strategy for their customers. Even if one root certificate is removed from the set of

¹⁸<https://www.eff.org/deeplinks/2011/08/iranian-man-middle-attack-against-google>

¹⁹<http://web.archive.org/web/20130926134541/http://turktrust.com.tr/en/kamuoyu-aciklamasi-en.2.html>

²⁰<https://www.openssl.org>

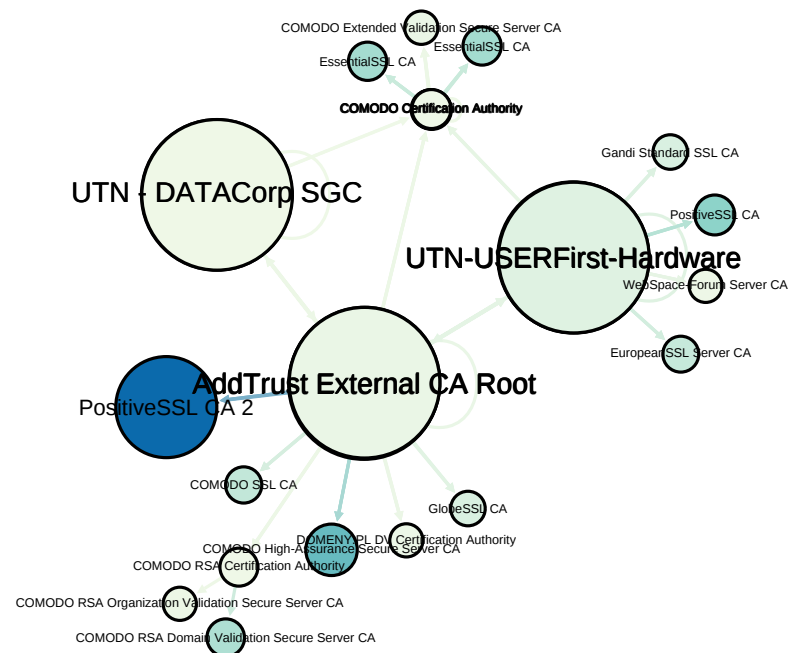


Figure 6.8: CA Topology of the Comodo CA.

trusted certificates, the customers' servers have other intermediate certificates to build a trusted chain. Although cross-certification is legitimate according to the X.509 standard, it implies a less secure CA, if the private key is lost.

Email Provider Strategies

Finally, the connections of the providers' email servers to other email servers were investigated. For this purpose, email accounts at various free email providers popular in Germany were registered and emails sent to prepared test email server. This revealed the connection details of the providers' outgoing connections. For this part of the study, Google Mail²¹, web.de²², GMX²³, Freenet²⁴, Yahoo Mail²⁵, Microsoft Outlook.com/Hotmail/Live²⁶, Apple iCloud Mail²⁷ and T-Online²⁸ were used.

The idea of this investigation was to find situations that a potential attacker with access to the connection between two email servers can exploit to read email. The attacker could just passively eavesdrop on the connection or actively perform a Man-in-the-Middle attack. To simulate an attacker that attempts to change cryptographic properties, the servers' TLS properties were changed to test the behavior of the email providers. In particular, the following settings were examined:

- Expired certificate (not security critical)

²¹<https://mail.google.com>

²²<https://www.web.de>

²³<https://www.gmx.de>

²⁴<https://email.freenet.de>

²⁵<https://mail.yahoo.com>

²⁶<https://login.live.com>

²⁷<https://www.icloud.com>

²⁸<https://email.t-online.de>

Table 6.5: Email provider strategies for connections to other email servers.

Provider	Untrusted Certificate	512-bit RSA Key	Anonymous Ciphers	Weak Ciphers	No STARTTLS
Google Mail	encrypted	no delivery	no delivery	no delivery	unencrypted
Web.de	encrypted	encrypted	unencrypted	unencrypted	unencrypted
GMX	encrypted	encrypted	unencrypted	unencrypted	unencrypted
Freenet	encrypted	unencrypted	encrypted	unencrypted	unencrypted
Yahoo	encrypted	encrypted	unencrypted	unencrypted	unencrypted
Outlook/Hotmail	encrypted	no delivery	unencrypted	unencrypted	unencrypted
iCloud	encrypted	encrypted	unencrypted	unencrypted	unencrypted
T-Online	encrypted	unencrypted	unencrypted	unencrypted	unencrypted

- 512-bit RSA key certificate
- Certificate with wrong CN
- Only anonymous ciphers (no certificate)
- Only weak or broken ciphers

Different security settings were found and thus more and less difficult situations to attack the email providers. All providers forward email to servers with invalid certificates and still communicate plaintext if a server does not support TLS at all. With one exception (Google), one can trick the email providers to send their data without any encryption. The results of the experiments are summarized in Table 6.5.

Invalid Certificates The main security properties of a certificate are a trusted issuer, the period of validity and the domain name matching the common name. All email providers ignored these features, completed the TLS handshake and submitted the data. A Man-in-the-Middle could provide this kind of certificate without any effort and thus see the data. In this study multiple certificates were used to check these properties. None of the providers complained about the often changing certificates, implying that certificate pinning is not used at all or at least not automatically for new or uncommon servers.

Default Cipher Suites All examined email providers use strong cipher suites in their connections, with one exception. Freenet offered, in addition to several strong cipher suites, three cipher suites with an anonymous key exchange. The anonymous key exchange methods do not use a certificate and therefore do not provide any authentication of the server. An attacker would love this situation, since his or her effort to read the email data is minimized.

Short RSA Keys To examine the use of short cryptographic keys, a certificate with a 512-bit RSA key was created. The email providers acted in different ways: Web.de, GMX, Yahoo and iCloud accepted the certificate, completed the TLS handshake and transmitted the email data. Google Mail and Microsoft Outlook/Hotmail rejected the certificate and continued with multiple unsuccessful retries. Freenet and T-Online rejected the certificate and closed the connection. Immediately afterwards, they opened another connection to transfer the email without using any encryption.

Weak Cipher Suites The behavior of the email servers when a server only supported weak ciphers was also checked. As mentioned above, the providers offer only strong ciphers in their client hello message. Therefore, the TLS handshake could not be completed. However, all providers except Google Mail reconnect after this failed TLS handshake and transfer the email without using encryption.

No STARTTLS In the last test, the STARTTLS command on port 25 was completely disabled. After the SMTP session is initiated, an email server offers a list of features it supports. STARTTLS is a means to encrypt the session after it is established. All tested email providers continued with the non-encrypted transfer of an email when this feature was disabled. Therefore, an attacker only needs to modify the SMTP command list provided by the server. Modern routing and firewall hardware, e.g., as provided by Cisco, has exactly this mechanism built in to inspect mail traffic²⁹.

Implications Considering the results presented above, an interesting situation happens: If an attacker is able to inject a few TCP packets into the connection, the TLS handshake is aborted, and the email provider reconnects and transfers the email without any transport security. Google seems to use a non-secured TLS policy in the sense that Google transfers email using TLS only, but does not insist on correct certificates.

Over the last years, several methods to avoid downgrade attacks in web browsers and other applications have been developed. Initiatives such as the SSL Observatory³⁰ and tools such as HTTPS Everywhere³¹ try to protect users against leakage of their private data. Email providers need to catch up and deploy similar security standards in their infrastructures.

6.2.4 Advice for Email Providers

The SMTP TLS landscape has major flaws across most email providers. In this subsection, some advice is given to email providers for obtaining secure configurations for their servers. The advice is meant to harden a TLS server at the cost of compatibility, which we think is one reason for weak configurations. In addition to the relevant TLS factors, measures to prevent simple downgrade attacks are suggested.

The following recommendations can be given to network administrators and email providers to harden their TLS configurations:

1. Update the TLS stack frequently.
2. Use TLS 1.0 (or higher) instead of SSL 3.0 (or lower).
3. Support strong cipher suites only.
4. Perform smart TLS certificate checks.
 - Time of validity
 - CA issuance
 - Key lengths
5. Accept/use TLS enabled email transfer only.

²⁹https://www.cisco.com/web/about/security/intelligence/asa_esmtp_starttls.html

³⁰<https://www.eff.org/observatory>

³¹<https://www.eff.org/de/https-everywhere>

The first advice is that the TLS stack of the system needs to be updated frequently to keep the system secure.

The second advice is that currently only TLS 1.0 and the higher TLS versions can be considered secure, since SSL 3.0 is affected by the POODLE attack.

The third advice is to support strong cipher suites only. Older cipher suites with MD5 hashing or 3DES encryption should not be used, since they are more likely to be broken in the near future. As with security updates for software and the TLS stack, email providers should pay close attention to the effects of new attacks on the list of secure cipher suites and remove insecure cipher suites accordingly³².

The fourth advice is to perform appropriate certificate checks regarding period of validity, CA issuer and key lengths and use a certificate pinning mechanism to detect bogus certificates³³. A list of the trusted certificates for every server must be maintained. When a TLS handshake is in progress, it can be checked whether the certificate has changed without revocation or whether it has expired.

The final advice is to either at least warn users and let them decide to use a potentially insecure connection, as done in web browsers, or decline non-encrypted email transfer, rather than just transferring email without encryption. One possibility to achieve this would be to set a special email header in the *Mail User Agent* (MUA) indicating that an email should only be delivered over encrypted links, with strictly validated peers or just anybody and any connection. Email should only be delivered over links with an equal or higher security rating than the one provided in the header. Although there is no guarantee that the server respects the user's wishes, if the big email providers followed this proposal and enough users set a flag for strict validation and encryption, smaller providers would be forced to use valid certificates and strong ciphers. Thus, unencrypted communication could possibly be eliminated after some time. It is also recommend to consider Ristic's TLS deployment best practices³⁴ for more details on TLS configuration hardening.

Furthermore, email providers are mostly left alone when it comes to tools for securing and testing their setups. Qualys provides a service for web server administrators to check their setup for common misconfigurations³⁵. This kind of service would be of great benefit for email providers. Standardized test infrastructures could also be utilized by central instances like CERT, the German BSI or large websites with lots of email traffic, such as Facebook, Youtube or Github, to automatically generate emails to MTA owners in case of security concerns. This, of course, means that every entity must support the common mailbox names as suggested by RFC 2142³⁶.

6.2.5 Conclusion

In this section, an empirical study of the security properties of SMTP over TLS within the German IP address space was presented. It was shown that even though many email servers support strong cipher suites, weak or broken cipher suites are still present. Furthermore, only few certificates provided by the email servers are valid (about 11%). The behavior of email providers differs significantly with respect to handling

³²<https://hynek.me/articles/hardening-your-web-servers-ssl-ciphers/>

³³https://www.owasp.org/index.php/Certificate_and_Public_Key_Pinning

³⁴https://www.ssllabs.com/downloads/SSL_TLS_Deployment_Best_Practices.pdf

³⁵<https://www.ssllabs.com/>

³⁶<https://tools.ietf.org/html/rfc2142>

various TLS configurations. Based on the results, practical advice on securing SMTP setups to avoid some of the identified issues was given. Unfortunately, an ultimate solution offering perfect security is simply not possible with the current PKI landscape. This topic is often discussed in the context of HTTP(S), but with email the problems may be even harder to fix, since decision making happens automatically without user interaction.

There are several areas for future work. For example, appropriate best practices are needed to ensure transport layer security of email traffic. A simple plain-text fallback as used by many providers significantly reduces the protection level of SMTP communication. Furthermore, validation and verification of peers and their certificates must be simplified. Finally, staying up to date with broken, weak and strong cipher suites is another challenging task, since there is no central entity giving advice for regular updates.

6.3 Hardening Server Systems

6.3.1 Introduction

External and internal intrusions are the most serious threats in computer systems connected to a network. Attackers exploit software bugs in core components on a target system to gain superuser privileges, allowing the attacker to take control of the attacked system. The rise of Cloud Computing aggravates the stated problem. Cloud Computing refers to both the on-demand provisioning of hardware resources in the data centers of public providers such as with Amazon's Web Services, and the applications delivered as services over the Internet, such as with Google's AppEngine. Offering access to remote compute resources is often referred as Infrastructure as a Service (IaaS). The resources provided as IaaS are platform virtualized environments, i.e. customers have access to their own virtual appliances running on shared physical resources.

To retain the control of the attacked system persistently, the intruders typically install malware in order to recover full control after reboot. The target system in this case is the virtual machine offered by the Cloud Computing provider. This kind of attack is commonly discussed as a *strong intrusion attack*, while temporary attacks between two operating system startups are referred as *weak intrusion attacks* [239]. The software toolkits that are installed within a strong intrusion attack are commonly called rootkits. Usually, weak intrusion attacks are used to place a rootkit on the attacked system.

These potential threats create the need for a new malware detection system. Providers need ways to ensure the security of their infrastructure and the systems of their customers. Having a flexible Cloud infrastructure also opens new possibilities to scale up and distribute malware detection software among several systems. Most end-host security solutions have a major, negative performance impact on the computer caused by huge signature-sets or complex detection algorithms. Cloud Computing can be beneficial here to decrease the slowdown and offload it to dedicated machines.

In this section, an approach that deals with malware detection and kernel rootkit prevention is presented. While the former deals with detecting malware traces during runtime in a safe and non-intrusive manner, the latter prevents rootkits from being installed in the operating system kernel. A Cloud-based intrusion detection system to recognize running malware is designed to run on virtual machine instances with a backend Cloud to distribute malware scanning operations between several backends. A flexible framework for a distributed security solution with a minimal overall resource footprint on the end host is presented. To detect well-known as well as yet unknown malware, a traditional signature check is performed and the prerequisites for a live system-call tracer are presented. Furthermore, the solution introduces an integrity check of authorized kernel modules to prevent rootkit installations via corrupted kernel modules. For this purpose, the operating system kernel is modified to load only previously cryptographically authorized kernel modules.

Parts of this section have been published in [13].

6.3.2 Problem Statement

A convenient way for an attacker to gain control over a compromised system is a rootkit. There are various types of rootkits available, e.g. application level rootkits that replace the original binaries with a fake binary containing a trojan horse or library rootkits

that replace valid library functions with malicious ones. The focus of our work is the *kernel rootkit*. It replaces/adds functions or device drivers in the kernel space of an operating system. Kernel modules in general enable upgrades to specified parts of a kernel to strengthen modularity of the operating system. There are two classes of kernel modules: permanent kernel modules, which are loaded at boot time and cannot be removed once they are running, and loadable kernel modules, which can be loaded and unloaded by the system at run time. Many kernel rootkits are designed as loadable modules or device drivers, since this is the easiest way to add new functionality to the core system. Thus, monitoring the loading process of kernel modules is indispensable to ensure that no malicious modules are loaded.

There are various ways to disable dynamic kernel module loading:

- In Linux it is possible to disable kernel module loading completely. While configuring a kernel, the administrator can set the `MODULES` option to `NO` and thus disables the complete kernel loading and processing mechanism. While this completely prevents kernel rootkits from loading, it also affects all legal modules.
- The technique of multiple secure levels is used in various BSD derived Unix operating systems. Any super user is able to increase the secure level. On the other hand, the only way to lower the secure level is via the `init`-process, a prototype user process that is only loaded during system startup, so the system has to be restarted. For example, FreeBSD [240], a widely used Unix branch, runs with four different levels of security.

Thus, it is possible to disable dynamic module loading either by disabling modules or via a higher secure level. In this case, one has to take the good with the bad: On the one hand, this avoids critical actions such as arbitrary changes of kernel memory through user programs (which, in fact, is performed by loading a kernel module). On the other hand, the concepts are very restrictive and forces users to compile and install the whole kernel instead of linking a single file. This step makes a reboot of the modified system necessary and interrupts running applications. Actually, for several applications (e.g. all mission critical applications), this is not a suitable solution.

While the previously stated problem applies to a greater extent to infrastructural machines, such as critical servers (e.g. DNS, DHCP), a Cloud provider should also be interested in keeping the VMs of its customers safe. Most Cloud vendors provide VMs with full root access, meaning that a user can mostly do whatever (s)he wants, including destroying the whole machine. Since Cloud Computing is about pay-as-you-go, this should not harm the vendor. Nevertheless, if a user (intentionally or not) executes malware, this could also affect the provider, e.g. a spam malware could abuse the outgoing bandwidth to send mass-spam mails. Thus, while granting root permissions to its customers, a provider still should be able to inspect the applications running inside its customers' VMs. Furthermore, (s)he should be able to take countermeasures if (s)he detects a security violation, such as running malware binaries. In the following section, we will present a Cloud based host intrusion detection system with a minimal resource footprint as well as hidden from the malware itself in the operating system kernel.

6.3.3 Related Work

Kroah-Hartman [241] has proposed to sign executables with a fingerprint. It is stored in an additional section of the commonly used executable linkage format (ELF). Furthermore, the technique of asymmetric cryptography is used to protect the fingerprint from malware modifications: A private key is used to encrypt the fingerprint stored in the ELF section, while the kernel linker decrypts the signature in order to compare it with the signature of the loaded file. A general problem is the kernel-level implementation of an asymmetric cryptography algorithm. There is no such implementation in most current operating systems. This is the reason why for this research the symmetric SHA256 hash algorithm was chosen.

A similar way of implementing a kernel rootkit prevention technique is used by Catuogno et al. [239]. They implemented a verification mechanism based on encrypted signatures stored in an additional section of an executable as well. In contrast to Kroah-Hartman, they did not address dynamically loadable kernel modules but executables in general. This is why they assumed that the support of dynamically loadable kernel modules should be disabled. This is not an appropriate assumption for the security of most applications.

In the NetBSD operating system, the Veriexec (verified execution) kernel subsystem allows users to monitor files and to prevent their removal, read/write access or execution if necessary³⁷. It implements four levels of strictness: A learning mode for configuration matters, intrusion detection and intrusion prevention mode, as well as a lockdown mode. Contrary to Veriexec, the proposed solution is specialized to protect the kernel from modifications by dynamically loaded modules. In this research a comparable database and fingerprints are used, but in contrary to the NetBSD kernel, the obsolete lkm (loadable kernel modules) architecture is not used.

King et al. [242] have classified three kinds of malicious services supported by virtual machine based rootkits (VMBR): Services not interacting with the target system (spam relays, DDoS zombies, phishing web servers), services observing data or events (keystrokes, network packets) using virtual machine introspection and services deliberately modifying the execution of the target system. They successfully implemented all of these types combined with a countermeasure against the redpill virtual machine detection mechanism through emulating an instruction, which is used to determine a difference between a real and a virtualized processor's interrupt descriptor table. The intrusion detection approach from this research cannot defend an attacked system once a VMBR is installed, nevertheless the proposed secure level mechanism is powerful enough to protect an endangered system from a VMBR installation by locking e.g. shutdown scripts used for Subvirt installation by King et al.

Garfinkel and Rosenblum [243] have described a virtual machine introspection based on an architecture that leverages the isolation, inspection and interposition properties of VMMs. Virtual machine introspection (VMI) describes a family of techniques that enables a VM service to understand and modify states and events within the guest. Beside this passive monitoring technique, active monitoring of virtual machine-based IDSes has been implemented as well [244]. Although they are facing the gap between the VMM's view of data/events and the guest software's view (which is called semantic gap), their modifications of the guest operating systems are detectable.

³⁷<https://www.netbsd.org/docs/guide/en/chap-veriexec.html>

CloudAV [245] is a software stack developed by Oberheide et al. It is meant to counter the problems single anti-virus solutions face nowadays with the increase of different malware and new exploit techniques. Instead of having just one AV solution per host, CloudAV uses multiple, heterogeneous detection engines. This approach is called 'N-version protection'.

The Automatic Malware Signature Discovery System (AMSDS) [246] has been developed by Yan and Wu. The fact that increasing numbers of zero-day malware take more and more time to analyze and the need to write signatures for them indicates that it is necessary to provide automatic signature generators. Moreover, the increasing size of signature databases and analysis techniques increase the processor and memory footprint on computers with installed anti-virus solutions. This can be countered by anti-virus software as a Cloud service, putting the workload of analysis and signature maintenance on dedicated machines. AMSDS has a small detection engine with a reduced signature set. This set of signatures can match a great share of malicious software through special treatment and preprocessing of the binary. Only if the much smaller AMSDS signatures cannot detect a suspicious file, it is sent to the Cloud anti-virus service for scanning with traditional anti-virus solutions. The automatic signature generation is very effective and space saving compared to classic signature generation. But these signatures can only detect binary executables loaded either from disk or network. An already running binary such as a service infected through an exploit is not covered by this approach.

Laureano et al. [247] have implemented some kernel introspection mechanisms into User-Mode-Linux. The authors gather information about the running system by inspecting the flow of the system calls made. Their IDS runs in two different modes: a mode for learning the regular behavior of a system and a so called monitor mode where anything unusual generates an alarm and suspicious processes are denied access to specific system calls. A similar system could easily be implemented within the framework presented here. Furthermore, in the proposed approach access to system call parameters is granted, enabling a more fine grained behavior analysis, while their approach just reports the system calls. Ignoring the parameters might lead to significantly more false alarms, since it can make a huge difference whether an *open* system call accesses a password file or just a new temporary file.

6.3.4 Design

In the sequel, the proposed solution to the problems stated above is presented. The proposal is based on the standard assumptions made in most other virtualization security architectures [243], [244]. The hypervisor is part of the trusted computing base (TSB). Since the focus is on infrastructural security, it is not dealt with attacks against the Virtual Machine Monitor.

Malware Detection

Contrary to a classic anti-virus setup, a Cloud specific design of a malware detection engine should run in a distributed manner and display some special requirements to ensure the security of the service provider's infrastructure as well as the customer's security. The communication paths and different software modules of the proposed design are shown in Figure 6.9. Any program run by the user (1) is executed in a virtual

machine in the Cloud. The kernel of this machine then passes all relevant information to a *KernelAgent* (2). The *KernelAgent* gathers all information by the virtual machines running on the Cloud resource and then relays them to the *ScanProxy* (3). The *ScanProxy* provides a front-end to the Cloud security analyzer services. At this stage, the proxy has to distribute the information to the different services, such as classic anti-virus software or behavior-based analysis solutions (4).

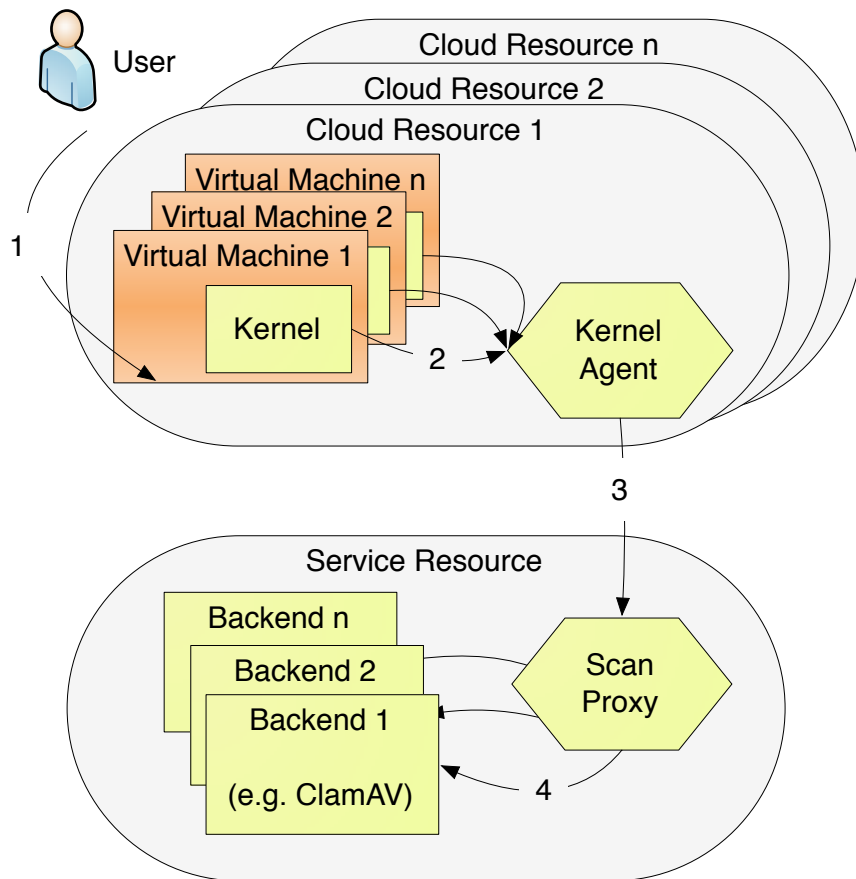


Figure 6.9: Malware scanner architecture

The kernel module is the primary sensor sitting directly in the running virtualized kernel of the guest machine. To avoid any security issues through the kernel module, it has very limited functionality. Its main task is to function as a logging relay and to submit any interesting activities to the *KernelAgent* for further processing. Valuable information include process creation or termination, system calls by these processes and the system call parameters as well as any binary files getting executed. This approach makes it very easy for the Cloud provider to maintain the system. The only component that has to be changed is the kernel. Thereafter, all operating system (OS) images that are provided by the customers can be booted using the modified kernel. Contrary to classic anti-virus solutions, no installation within the OS image is necessary, which means the additional security provided by the OS is completely transparent to the customer. Moreover, the customer has full control over his or her OS image. No matter what the customer does with the image, (s)he cannot break or deactivate the malware

detection system.

The kernel modules should intercept any executable before it is running and submit it to its host agent. This is the way classic anti-virus hooks grab an executable before loading it into their scan engine. They check every executable through static analysis. Applying static binary analysis might not always be the best way to ensure security, especially when confronted with unknown, new malware. Nevertheless, it still should be part of any malware detection solution. Using this approach, it is easy to take advantage of all the existing anti-virus software. A requirement for any executable analysis is the binary image of the file itself, and for identification purposes, the filename must also be transmitted.

Process Life Cycle, System Calls Monitoring a process with respect to its system calls throughout its lifecycle can be a valuable source of information when looking for common patterns in malware behavior. By relaying this information live, not only encrypted executable images and obfuscation, but also in-memory injected malware through an exploit can be analyzed. System calls make it easy to spot specific file accesses or socket operations, such as transmitting data back to an attacker. The relevant information includes the system call, its parameters, return values and the program that made the call.

KernelAgent This part collects all the data from the VM kernels running on the host system. This information should then be relayed to the *ScanProxy*. Since there is no other logic involved in this piece of software other than the configuration of what has to be sent to whom, there is almost no need to touch an installed system. To increase the performance, caching of messages and later on responses is implemented. This is especially interesting for classical executable image analysis. While starting-up several virtual machines, the same executable is run several times. These often called binaries include e.g. system services. Submitting and analyzing the executable at every initiation/run costs CPU time and also increases network traffic. This can slow down the start-up time in a feedback based intrusion prevention system significantly.

Since both groups of information (binary and system call related) have different requirements, splitting up the *KernelAgent* into two separate servers makes sense. One is a UDP-based system call forwarder, the other one should receive binaries and forward them. The binary executable relay must not save any executables to the hard disk. Otherwise, there is a chance of an infection happening on the host system in case of malware.

ScanProxy This component gathers all available information from the hosts and distributes it among the registered scan engines. For each incoming packet containing a system call, one or more receiving scan engines can be used. The proxy then forwards the packet to the registered receivers. It could also act as a global log and caching proxy for the complete Cloud. Every new scan engine being a system call analyzer or a classical anti-virus scanner can be enlisted here once or even several times for redundancy purposes. The proxy does not need to have much more logic than the above to keep the system as easy to manage and immunized as much as possible. More code complexity means more space for failure through attacks.

Scan Engine and Executable Analysis Considering the previously described framework, several possible scanning backends can be implemented. They can generally be categorized as process behavior based or executable binary based, such as a classical anti-virus solution, e.g. ClamAV³⁸. Every incoming executable has to be placed in a separate container on the hard disk and then has to be analyzed. The received binaries must not be executed, otherwise the security of the scanning computer can be compromised in case of an infection. By registering several different anti-virus scanners with wrappers, an increased level of security can be achieved. This helps to minimize the vulnerability window that exists between the discovery of a new malware and the release of the signatures by the anti-virus vendors for their products. To process events such as systems calls, a backend like the software of Wagener et al. [248] can be used with minor modifications. The underlying concept of their approach is that even new malware shares common behavioral similarities to already existing malware. By finding these similarities in behavior graphs, even yet unknown malware can be automatically detected. While Wagener et al. perform system call analysis ahead-of-time in a secure execution environment, modifications should easily be possible to enable on-the-fly detection.

Kernel Rootkit Prevention

Instead of disabling kernel module loading completely, the BSD secure level concept is favored. It allows module loading before raising the secure level. The following subsection describes the process of kernel rootkit prevention by loading authorized kernel modules only. It is distinguished between two modes, describing the state of the secure level. If the secure level is lower or equal than 0 (which is the default for single user mode), it is called insecure mode; if the secure level is set to 1 or higher (kernel memory is read-only, file system might be read only), it is called secure mode. Furthermore, adding a module to the internal list is called mark/unmark as authorized.

To prevent kernel rootkits, it is distinguished between safe and unsafe kernel modules. In secure mode, it is only possible to (un-)load authorized kernel modules. It is not possible to load other modules, especially any kind of malware. All authorized modules are kept in a list that resides in read-only kernel memory. The latter is needed to prevent that an attacker could simply modify the list to mask a rootkit as an authorized module. Each list entry contains the following information:

- A human-readable description of the kernel module
- A unique cryptographic hash of the kernel module
- Some internal kernel structures to indicate whether the kernel module is currently loaded

The implementation uses a generated SHA256 hash to provide a unique key for each module. To authorize kernel modules, a userland program has been developed to add or remove kernel modules to the mentioned list through a system call. This system call refuses execution if it is called without root privileges. Optionally, all dependent modules could be added as well. Any operations on the list can only be made while the system is in insecure mode. A convenient moment would be the initial system setup

³⁸<https://www.clamav.net/>

before it is actually connected to an external network. While the system is running in insecure mode, the userland program is able to mark/unmark modules as authorized. The list, where the marks are stored, uses transient storage, i.e. the list is initially empty at system start-up time.

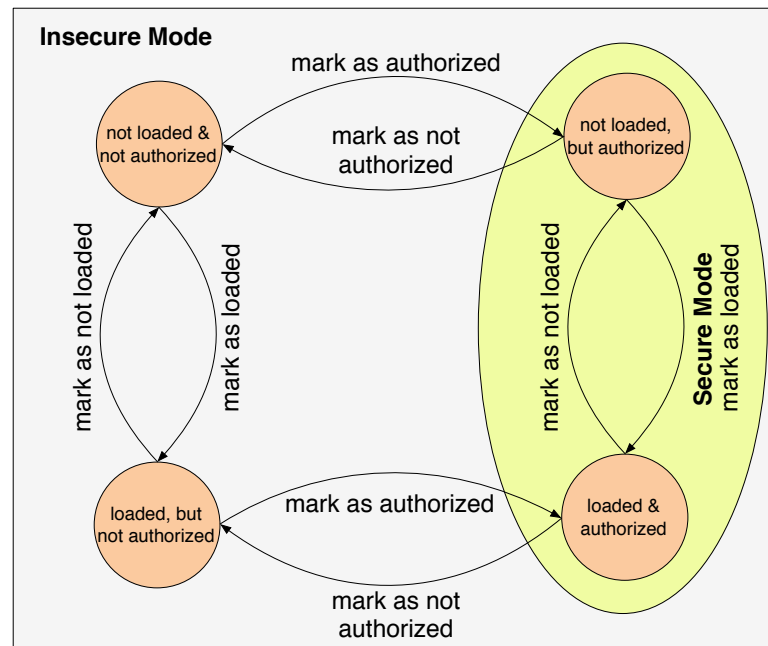


Figure 6.10: Authorized module loading state transition diagram

Figure 6.10 shows the possible modifications of a list entry. By default, a module is not loaded and not authorized. In insecure mode, a user can mark a module as authorized and thus is able to load it later when the system is in secure mode. All kernel modules that are loaded during the boot process (e.g. the ACPI subsystem or device drivers) are not authorized. Consequently, they have to be authorized before the system is switched to secure mode. Otherwise, they would work as expected, but unloading would not be possible (which might not be necessary, especially if it is a core component). Once the system is in secure mode, only authorized kernel modules can be loaded.

The main features of this process are encapsulated in the dynamic module loading process to check whether a module is marked as authorized or not. To authorize a module, an authorization function has to open the module file, hash its content and search for matching hashes in the list. If the authorized-flag of the corresponding list entry is set, the module is allowed to be loaded. The unloading process is handled by another function that checks if the module is already loaded. Consequently, we do not have to hash the module again. Every loaded module is equipped with a unique pointer, representing the module. This pointer is used to find the correct module in the list and to decide whether to unload or not. Finally, unloaded modules must be marked as not loaded in the list.

6.3.5 Implementation

Like the operating system kernel (which in this case is the DragonFly BSD kernel, version 2.5.0), the kernel part of the malware detection module and all parts of kernel rootkit prevention have been written in ANSI C.

Malware Detection

To tap into the relevant parts of the kernel, some static hooks are installed. These hooks redirect or copy valuable information from kernel functions such as *execve* to an extra function that passes this information on to the *KernelAgent*.

Process Related Information Getting all process related information requires the addition of several hooks to the VM kernel. A hook is installed in the function that adds new processes to the kernel's process list and assigns a new PID to them. The list is a linked-list used to keep a global list of all running processes. Another hook that is called at the end of a process lifetime works in a similar fashion. This routine is called by the kernel's *exit1* function to remove a process from the global list of running processes and add it to the list of dead processes. This list is an in-kernel linked-list containing all processes in the *ZOMBIE* state. This means that they are about to be removed from memory and are done executing.

The system call hook is called from within the VM's *syscall2* function. It is executed immediately after the processing of the real system call. Getting called after the execution of the system call has the advantage that some parameters that are passed on empty to the kernel and are filled during execution can get their content inspected. This is, for example, the case with the *open* system call that has a buffer as its parameter for reading bytes from a file descriptor.

The challenging part here is getting the parameters. They are passed to the system call function without providing any type-information, except a memory reference. For the kernel, there is no need to know this type-information, since the corresponding system call knows what type its parameters should have. As part of this approach, an extra file holds a list of all system calls and their parameter types. Additionally, the error code as returned by the actual system call is provided for analysis purposes. This has the advantage that the data flow can be recorded, such as the returned file descriptor from an open call and later on any *read* system calls to this file descriptor.

Executable Loader The binary loader hook is placed in the *kern_execve* function of the VM, which is the actual place of execution and not the system calls' first entry point, *sys_execve*. To avoid unnecessary calls to the logging hook, it is only called after *exec_check_permissions* has successfully returned. After this call, it is certain that the executable is valid and has the appropriate permissions. The logging takes place before the first page of the executable gets mapped into the memory and is executed. In a feedback-based Intrusion detection system it would still be possible to stop the execution at this stage, should the binary be infected with malware. The whole binary is then submitted to the *KernelAgent* using TCP.

Communication from the VM to the Host System To keep the protocol overhead as small as possible and be as responsive as necessary, a simple protocol is implemented

by using UDP in the kernel. Approaches based on TCP would have brought up some additional delays, which is a problem when monitoring realtime events such as system calls.

KernelAgent The *KernelAgent*'s main task is to collect the data from all virtual kernels running on the machine and forward it to its *ScanProxy*. This part is implemented using the Python scripting language. Whenever a new packet is received, a background thread is started to process the received packet. This implies that it is parsed and then the whole packet is sent forward to the configured *ScanProxy*.

ScanProxy This software module is similar to the *KernelAgent* on the receiving part and is also written in Python. Instead of one configured receiver for relaying, like in the *KernelAgent*, there is a list of receivers. This list can be configured for each entry to relay only specific types of traffic (e.g. only NEWPROC, ENDPROC and SYSCALL) or any traffic for a catchall or logging daemon. Due to this fine grained configurability, the incoming packets must be inspected and checked against the list of receivers to ensure that every receiver obtains only events that have been subscribed for.

For scanning executable files with an anti-virus software such as ClamAV, a TCP variant of the *ScanProxy* has also been written. Just as within the UDP *ScanProxy*, a list of receivers/backends can be configured. All incoming binaries are relayed to them. The *ScanProxy* only keeps the executable's data in memory, nothing gets written to the hard disk. This backend checks incoming binary files with ClamAV for known viruses. Incoming files are received over TCP connections to ensure that the received binaries are complete and in-order. As in the *KernelAgent* and the *ScanProxy* the name of the executable is also submitted. Every received binary is saved in a temporary quarantine folder, where it is scanned. After scanning is done, the file gets deleted to ensure security of the backend system.

Kernel Rootkit Prevention

All main communication between userland tools and kernel is handled by a newly introduced system call, e.g. add a new module to the internal list is done by sending the required information via the defined interface. As mentioned before, the internal list has to hold any information about a module. For example, a module can just be authorized, not loaded, or it can be loaded but not authorized. Therefore, our list contains one entry per module. The state is held in flags or implicitly by pointers being not empty.

Every generated list entry holds a unique key. In this implementation, the generated SHA256 hash is used to provide a unique key for each module. The longer the resulting hash value is, the more secure is the corresponding algorithm regarding brute force attacks. Thus, SHA-256 is a good tradeoff in terms of security as well as memory usage and performance.

The identifier is used to hold the human readable description of that entry. A *linker_file* pointer points to the corresponding linker file kernel structure. This is a convenient way to map a loaded module to the generated hash without doing changes inside the existing kernel structures. To prevent the list from being changed while system is in secure state, the functions responsible for mark-and-authorize a module are not callable

in that case. After switching to secure mode, only authorized modules can be loaded or unloaded - there is no way to authorize kernel modules retroactively.

Usually a userland tool is used to load modules during runtime. This utility directly uses the system call *kern_kldload*, which basically implements dynamic module loading. After a basic permission check, the main module loading, depending on the binary format, is performed. Those formats are compiled into the kernel and cannot be changed dynamically. The common format is the Executable and Linking Format (ELF). Each of the functions above verifies the secure level at first and interrupts loading immediately if the system runs in secure mode. Figure 6.11 shows a flowchart of the module loading process (changes drawn in dashed lines).

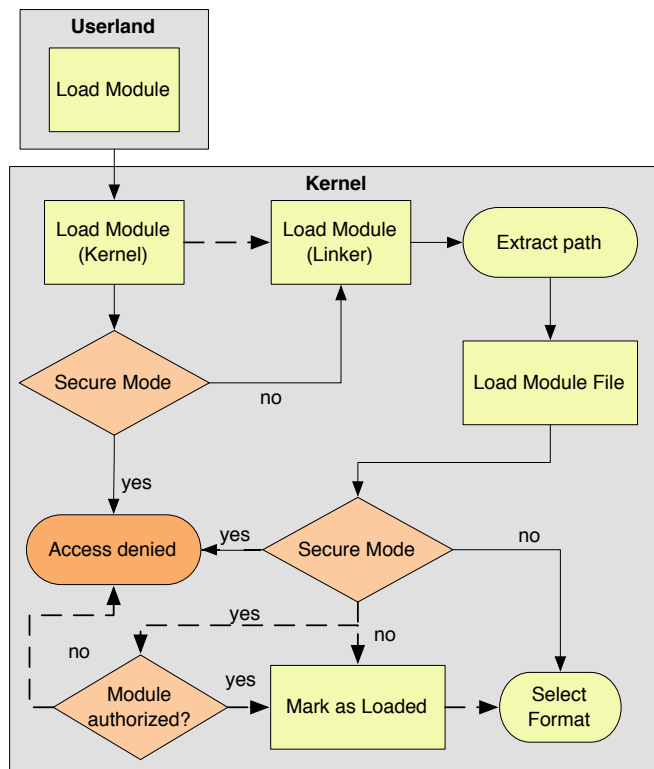


Figure 6.11: Module loading activity

The first task to authorize a module is to open a virtual node, identified by the given filename. A virtual node is an entry in the virtual file system (VFS), which is an abstract layer on top of the physical file systems. The function has to open the virtual node already in this early stage of the loading process. Later on, one could reuse the provided, convenient functions to read a file from kernel, but from the security perspective this would be too late. Thus, the more complex way through the VFS layer has to be taken. As a consequence, the virtual node, pointing on the module file, is opened twice during the loading process. As this is not a time-critical job, it is negligible. If the virtual node is opened without errors, the module is read. Otherwise, the function aborts with an error message. Since the read bytes can be added to the hash algorithm successively, the memory usage by reading data piecewise using the same buffer each time is reduced. This data can be used to generate the hash key. If the internal list contains the generated hash key, the module is marked as to be loaded, otherwise it is not and the appropriate

6 Secure Cloud Systems

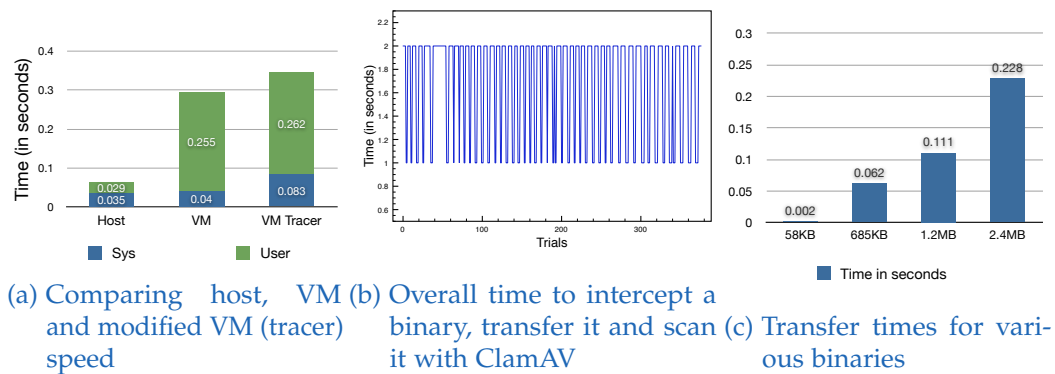


Figure 6.12: Malware Detection Benchmarks

permission denied error is returned.

Authorizing a module within the unloading process is less complex, because the used data structures by the unload process contain a file pointer that is also registered in the internal list if the module is loaded. If the module is authorized, it is unloaded. Otherwise, unloading is not permitted.

6.3.6 Experimental Evaluation

This subsection focuses on the performance and a qualitative evaluation of the developed prototype malware detection and kernel rootkit prevention system. All tests were performed on two 2.53 GHz Intel Core2Duo CPU, 4 GB MB RAM running DragonFlyBSD 2.5 connected with Gigabit ethernet.

Malware Detection

Since the main modifications to the VM kernel happened in the process and system call handling code, measuring performance impact is best done by spawning several processes and by performing rapid system calls, thus data or process intensive tasks are not relevant for the benchmark. A test case that queries the kernel for network, user and other arbitrary information is executed 50 times, and the average run-time is calculated. The results of the benchmark are presented in Figure 6.12a. The lower bars, named *sys*, indicate the time spent executing system calls on behalf of the executed program. The upper bars, named *user*, represent the time spent doing calculations, iterations or generally spoken actions in userland. The diagram clearly shows that the host operating system easily outperforms the virtualized kernels. Even though the time spent executing system calls is nearly identical between host and the VM kernel, the time spent in userland is much more compared to the time when running on the host directly. Enabling the tracer functionality of the VM kernel doubles the time spent in the kernel, but the userland portion stays constant. Since the in-kernel time for system calls is so low compared to the total execution time (0.06 seconds), this impact on the performance can be neglected.

Figure 6.12b shows the measured time needed to intercept a 8 KB binary in a running VM with the *KernelAgent*, transfer it over the network and scan it with the ClamAV engine. Over 350 trials were conducted to get a robust mean, which is 0.5 seconds. The oscillation of the graph is due to the fact that we could only measure with a wall

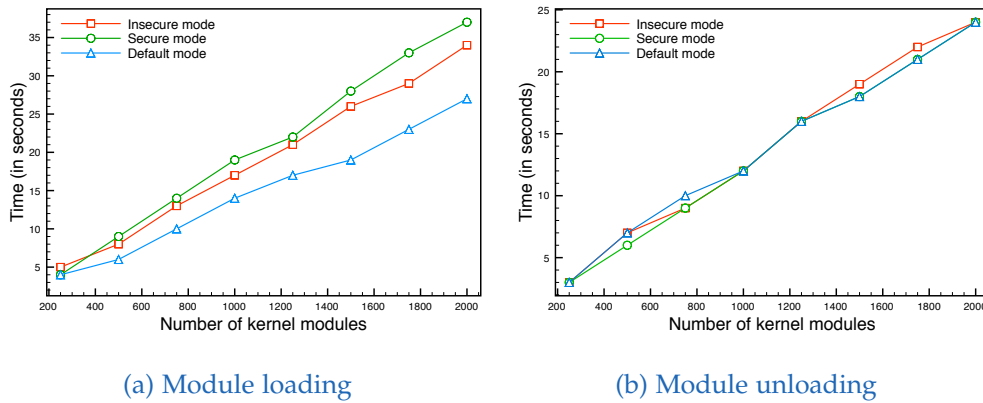


Figure 6.13: Module (un)loading overhead measurement

clock, which in this case works with Unix timestamps (seconds since the epoch). The measured overhead of 0.5 seconds before the actual execution starts is negligible in the described Cloud environment, since most jobs will be long running computational jobs. Furthermore, the use of caching techniques will even reduce the overhead, as every (unchanged) binary is only scanned once. Figure 6.12c shows the times needed to transfer various binaries of different sizes over the developed middleware between *KernelAgent* and the *ScanProxy*. Multiple measurements with different binary sizes representing different types of malware (the average file size of the standard system binaries is about 1.2 MB) were conducted. For binary 1 (58 KB), the average time is 0.001 seconds, for binary 2 (685 KB), the average time is 0.06 seconds, for binary 3 (1.2 MB), the average time is 0.1 seconds, and for the biggest binary (2.4 MB), the average time is 0.2 seconds. Thus, the transfer time increases with the size of the binary.

Kernel Rootkit Prevention

To measure the module loading overhead, a script was written that cascades module (un-)loading. Since the main overhead is due to hashing the modules, a proper average module size had to be chosen to get realistic results. By examining the standard kernel directory, 64 kByte turned out to be the average size of the kernel modules. In order to be a bit ahead, 100 KB sized kernel modules were created for testing. To be able to measure the correct time overhead, the modules were loaded between 250 and 2000 times.

As shown in Figure 6.13a, there is an overhead in every measurement. The module hashing causes the overhead during every module load. Loading modules either in secure or insecure mode (with enabled kernel protection) takes more time than loading modules in the default mode (no protection and a stock kernel). This is due to the fact that the kernel rootkit prevention technique has to iterate over the internal list to validate a module. Thus, there is an additional linear effort. Nevertheless, module loading is not a time critical job and the average number of loaded modules should be much lower than in the tests. In the case of 250 loaded modules in the generic kernel, a module needs 0.016 seconds on the average to get loaded. In a kernel with rootkit prevention it takes 0.02 seconds on the average. This is more than 1.25 times longer, but still is not a large delay. If the system is running in secure mode, loading a module

will consume more time, because there is one additional list iteration involved in the loading process. Generic kernels are not even able to load modules during secure mode. The measured overhead for module unloading is shown in [6.13b](#). Contrary to the loading process, the unloading process is less time consuming. There is no noteworthy time difference regardless which kernel mode is used.

6.3.7 Conclusion

In this section, an approach for combined malware detection and rootkit prevention in Cloud Computing environments was presented. All running binaries are intercepted by a small, in-kernel agent and submitted to one or more backend units where the actual classification process happens. Furthermore, live-scanning of all binary system calls is performed to detect yet unknown exploits or malware. Due to the in-kernel nature of the agent, it is completely transparent to the user as well as to malicious binaries trying to detect any countermeasures. The distributed architecture allows a good utilization of existing Cloud resources and the connection of different analysis engines.

While the detection rate of malware and anti-virus scanners has steadily improved within the last years, its still not a fool-proof solution against recent exploits like zero-day exploits. Many successful attacks lead to the installation of a kernel rootkit to gain permanent control over the target machine including the possibility to get access at later times and misuse the machines as an attack platform. Consequently, the proposed solution is a modification of the in-kernel loading process. Only authorized and thus trusted kernel modules are allowed to load during runtime. Loading of unauthorized modules is no longer possible.

There are several issues of future work. For example, the malware detection engine currently implemented provides a solid foundation for a flexible Cloud specific anti-malware solution. In a second step, it would be desirable to change the software stack from just a detection engine to a bidirectional intrusion response engine capable of isolating and terminating malware binaries in real-time. For the rootkit prevention solution it would be desirable to bring asymmetric cryptography into the various operating system kernels to be able to use signatures instead of cryptographic hashes. Finally, the possibility to manage the in-kernel black- and whitelists (or a central signing key) could be realized by a central instance in the Cloud Computing environment. For this purpose, various parts of proposed infrastructure could be reused to achieve this goal.

6.4 Reactive Realtime Cloud Infrastructure Monitoring

6.4.1 Introduction

In recent years, the number of security attacks performed on computer systems connected to the Internet has increased significantly. Several security monitoring solutions such as Intrusion Detection/Prevention (IDS/IPS) and Security Information and Event Management (SIEM) systems have been developed as a response to the rising number of threats. To work effectively, these systems heavily rely on meaningful log data generated by different components involved in the setup. Automatically parsing these human readable logs costs considerable amounts of time given the verbosity of the information collected. Furthermore, the more complex an attack is, the harder it gets to detect it using simple signature-based approaches, since they typically support only simple count-based rules or simple regular expression matching rules for particular attacks. Supporting more complex rules, on the other hand, usually suffers from keeping up with high event rates and processing them in real time.

In this section, a new approach to detect, analyze and handle security anomalies is presented. The anomalies include both known and yet unknown security vulnerabilities with a particular focus on systems based on operating system virtualization, such as Infrastructure-as-a-Service Cloud computing systems. Hence, a novel SIEM system especially for virtualized computing resources is proposed.

The proposed approach utilizes sensors deployed on different layers of a virtualized computer system. Layers range from the hypervisor, also called virtual machine (VM) monitor, and the operating system of the VM (kernel and userland) to any kind of application runtime environment, such as a web-application container, to continuously report all relevant events. The combination of out-of-VM monitoring using VM introspection [244] and in-VM monitoring opens up opportunities to eliminate false positives through double bookkeeping.

To facilitate horizontal and vertical correlation and aggregation of monitored events, Complex Event Processing (CEP) is used. Continuous queries on event streams perform cross-layer monitoring of events to detect security anomalies. These queries can trigger different actions in any of the virtualization layers to repel attacks or isolate breached components. Therefore, attacks can be stopped before they are successful, or further breaching of a compromised VM is prevented.

The core functionality resides in a hardened, trusted VM responsible for analyzing sensor data. All sensor data is sent to this VM for further processing, and actions are triggered by this VM if necessary. To increase the speed with which events are processed by queries, queries are distributed across multiple, different CEP engines optimized for specific query operators. Furthermore, a global event log called Event Store is maintained, gathering all events and providing the possibility for offline learning and what-if analysis. Analysis results produced offline from the Event Store can then be fed into the online query rules.

Thus, the proposed approach is designed to provide *security monitoring in a box*, meaning that every physical server hosting multiple virtual machines has a dedicated virtual machine (VM) responsible for performing all tasks required for monitoring this server. Therefore, monitoring a very large number of physical servers is possible, since each additional physical server gets an additional dedicated VM for security

monitoring. The particular focus of this design is on systems based on operating system virtualization, such as Infrastructure-as-a-Service Cloud computing systems. Hence, we propose a novel SIEM system for virtualized computing resources.

This research makes several contributions to advance the state-of-the-art:

- A novel set of minimally intrusive sensors with a reduced attack surface enabling live gathering of sensor data is presented.
- A novel approach to optimize the performance of continuous queries by utilizing a federation of heterogeneous CEP engines is presented.
- A novel historical data store, optimized for fast event logging is presented. It is designed for fast sequential writing and applies a bulk-loading technique for almost no-cost index creation.
- A novel query language for specifying complex monitoring rules compared to classic IDS and SIEM solutions is presented.
- A secure execution and deployment environment for sensor and action scripts is presented.

The research focuses explicitly and solely on the components of the infrastructure required for performing security monitoring and not on the improved security through advanced attack signatures or anomaly detecting queries. Some examples of queries to detect simple security incidents are presented in this work, but developing new attack signatures or automatically generating queries based on past behavior are beyond the scope of this work.

Parts of this section have been published in [14], [15].

6.4.2 Related Work

CEP Engine. The main features of the CEP engine used in this system are that it is a federation of different CEP engines and that it utilizes a query index. While the former is used to optimize the performance of single queries, the latter is used to ensure scalability with an increasing number of queries. Federations of data management systems are not novel. Federal database systems [249] have been intensively studied in the past. However, the main focus of federated database systems is the integration of distributed and potentially heterogeneous databases and not performance optimizations. The same is true for MaxStream [250], the only existing federal stream processing system we are aware of. MaxStream extends SAP MaxDB, a federation engine for database management systems (DBMS), by the ability to also integrate data stream processing systems (DSMS) and has a strong emphasis on the cooperation between DBMSs and DSMSs. Performance optimizations among a federation of different DSMSs are not studied by MaxStream. The cyclops platform [251] is similar to the approach used in this work, but focuses on federations consisting of different classes of data management systems rather than on federations consisting of different implementations of the same type of data management system. In particular, Cyclops optimizes the performance of continuous aggregation queries among a federation consisting of a centralized stream processing system, a distributed stream processing system and a distributed batch processing system. Based on the definition of the window, one system class is superior to the others from a performance point of view. For instance, when the window is a window that slides from event to event, a centralized stream processing

system performs best. When the window jumps with big distances, a distributed batch processing system performs best. Similar to the approach from this research, Cyclops selects a target system class via a classifier that has been created on basis of performance benchmarks. However, Cyclops focuses only on continuous aggregation queries that are an atomic unit and not on complex queries consisting of several operators that can be distributed. The latter problem has been studied by StreamCloud [252]. StreamCloud partitions operator graphs into arbitrary subgraphs and distributes the resulting parts across a distributed streaming infrastructure. The partitioning is based on a simple but effective strategy. For each stateful operator, an operator graph is cut. Therefore, costly stateful operators can be optimally distributed with respect to load balancing, utilization of the infrastructure and performance. All relatively cheap stateless operators are then automatically distributed together with their associated stateful operators in order to reduce communication overhead. However, StreamCloud focuses only on infrastructures consisting of a single type of streaming system and not on heterogeneous federations.

Recently, the BE-tree [253] was introduced for indexing sets of queries. The BE-tree is a dynamic query index being superior to all competitors proposed so far (e.g., Gryphon [254], k-index [255]). Therefore, the BE-tree for the Analysis-VM of the system was reimplemented. However, the system does not need a fully dynamic query index since index updates are very infrequent in comparison to the event rates. Therefore, a bulk-loading technique for BE-trees was developed to perform several optimizations to further increase the performance of BE-trees.

Sensor and Actor Frameworks. The foundation of every decision or rule-based system are the sensors that deliver the data being used. There is a variety of data sources developed for different purposes such as alerts, performance monitoring and intrusion detection. Alert-based systems such as Nagios³⁹ are organized in a host- and service-based manner, meaning that attributes of the host and running services are monitored by running specified commands and using specified polling values. Typical events reported are the information whether or not a service is vital and how this information was determined, as plaintext printed to stdout. Countermeasures are not part of the system, since the main purpose is the notification of responsible administrators. Performance monitoring frameworks such as Ganglia⁴⁰, Munin⁴¹ and Scout⁴² place agents on the hosts to be monitored. Sensors are scripts in any language the system supports, executed by these agents. In contrast to the polling-based approach, intrusion detection systems collect data continuously. Snort⁴³, a network intrusion detection system, collects network packets and uses regular expressions to detect malicious activity or specific attack patterns. Ossec⁴⁴, a host-based intrusion detection system, performs log file analysis, file system monitoring, root-kit detection (Unix) and registry integrity checking (Windows). While data is being collected by agents, the analysis is performed by a central manager using regular expressions. Ossec uses pre-shared keys for encryption and *zlib* for compression of the data sent via UDP and can trigger single commands at monitored sites or the server. Despite the fact

³⁹<http://www.nagios.org/>

⁴⁰<http://ganglia.info/>

⁴¹<http://munin-monitoring.org/>

⁴²<https://scoutapp.com/>

⁴³<https://snort.org/>

⁴⁴<http://www.ossec.net/>

that one could have utilized all of the mentioned frameworks with little effort in the proposed reactive security monitoring system, the solution proposed here already offers minimally invasive sensors on all layers of a virtualized computer system collecting live data and event-based data, secure communication optimized for fast transmission and signature-based deployment and execution of actions. Forget et al. [256] have proposed a design for a client/server infrastructure for monitoring client machines in which they describe how sensors should be designed to be able to collect data for long-term analysis. In contrast to this research approach, data is written to a data server before it is analyzed, but the general design principles of sensors, namely being independent, minimally invasive and running in a least privileged manner, are similar what is done in this work. DACSA [257] presented by Gionta et al. is a decoupled architecture for cloud security analysis performing out-of-VM analysis without interfering with the guest system. To achieve this goal, the authors create copy-on-write snapshots of running VMs and analyze the memory of these snapshots using forensic methods and the ClamAV Anti-Virus software. Through this technique, the overhead for analysis can be reduced at the cost of not being able to analyze real-time data. Srinivasan et al. [258] have proposed a system for fine-grained out-of-VM process execution monitoring by moving the process to an analysis VM and redirecting kernel level operations back to the guest system using two techniques called out-grafting and split-execution. In this way, it is possible to use userland analysis tools such as *strace* to analyze processes from another guest system. While this is useful for malware analysis where the tools for inspection have to be hidden, the goal in this work is to prevent that any harm gets done.

Stream Data Stores. Storing streams is a great challenge due to their huge data amounts and high data rates. In contrast this work, recent systems are either based on relational database systems or distributed key-value stores. Golab et al. have proposed a data warehouse solution called *DataDepot* [259] to store streaming data. The authors use a two-tier approach: at first, data is simply written to disk (i.e., into log files). Afterwards, an ETL query transforms streaming data from the raw data sources into a relational representation and loads them into the data warehouse. Apart from these raw tables, *DataDepot* supports derived tables constructed via SQL queries and data partitioning. In contrast to this approach, *DataDepot* uses a relational database as its underlying storage and achieves a throughput of only about 10 MB/s. *Tidalrace* [260], the successor of *DataDepot*, pursues a distributed storage approach and reaches data rates of up to 500.000 records per second, which still does not compete with the centralized approach in this research. *LogKV* [261] utilizes distributed key-value stores to process event log files. The authors assume a scenario where many different log sources (machines) with different event formats are connected to *LogKV*. A central coordinator node maintains the meta-information about the log sources and is responsible for the distribution of the incoming data to a set of worker nodes in the same data center. Every worker node is divided into two components. *TimeRangeKV* represents the final storage layout and manages all data belonging to its dedicated temporal partition. Since a single machine may be overwhelmed with the incoming event data, several *IngestKV* instances of multiple worker nodes first ingest incoming events to support high data throughput. Afterwards, *IngestKV* shuffles the log data to the desired worker node with the dedicated *TimeRangeKV* instance for its time partition. In their experiments, the authors achieved a throughput of 28 MB/s log ingestion bandwidth per worker node

consisting of an Intel Xeon X5675 system with 96 GB memory and a 7200rpm SAS drive, which are connected via a 1 GB/s network. Deri et al. [262] have presented a lightweight and fast time series database based on the embedded BerkeleyDB. The authors store time series as BLOBs to reduce the number of transactions, and therefore to increase data throughput. Similar to this work, the authors use a LZ based compression library for loss-less data compression. In contrast to this research, the authors assume all streams of the same database to have the same equidistant time intervals. LogBase [263] is a distributed, write-optimized store with transaction support. It uses a log-structured data base, and like HBase⁴⁵, it uses super-nodes that are responsible for a group of adjacent nodes. LogBase stores all data in a single unordered data log that is located on a distributed file system. The log approach has append-only semantics. This offers sequential write performance on disks. In contrast to this work, LogBase is designed as a general-purpose database, also applicable for media data like photos. Due to the fact that LogBase is based on Hadoop⁴⁶, it is not able to keep up with the real-time demands of a storage for event systems. The authors use an index similar to B^{link} -trees, augmented with compound keys (key, timestamp) to index the data in an in-memory multi-version index. In the proposal of Wang et al. [264], LogBase acts as storage layer for a lightweight indexing approach. Similar to this approach, the authors utilize the assumption that successive observational values are similar. Therefore, they index (min, max) intervals for each disk page within B^+ -trees and interval trees to speed up query performance. These (min, max) values cover a small amount of space compared to the raw data and therefore are assumed to be kept in memory.

6.4.3 Architecture

The goal of the proposed reactive security monitoring system is to detect attacks based on the analysis of sensor data. Therefore, it is crucial to gather as much sensor data as possible. Theoretically, every event that occurs in one of the different layers of a virtualized system can be an indicator for an anomaly: for example, established network connections, creation or termination of processes or even user or process activities beyond regular working hours.

The decision about what is a normal or unknown system behavior cannot be made by the sensors of a monitored environment. Instead, a CEP engine is responsible for processing all the informations sent by the sensors and is then able to decide what can be viewed as normal system behavior. Through dynamic deployment of further sensors, it is possible to eliminate false positives and verify findings.

Collecting large amounts of data leads to challenges regarding the transport, the analysis and the storage of sensor data. In addition, transitions from knowledge derived from data considered as normal to detection rules and appropriate actions have to be considered.

The architecture of the proposed system (Fig. 6.14) consists of a secure and trusted virtual machine (called Analysis-VM) where the main analysis component is located, sensors that reside in every layer of the virtualized computer system and actors that execute actions. More design details for all components are given below.

⁴⁵<http://hbase.apache.org/>

⁴⁶<http://hadoop.apache.org/>

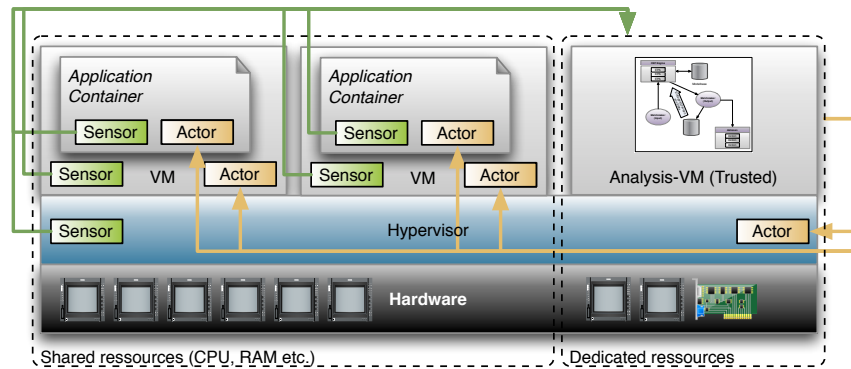


Figure 6.14: Sensors, Analysis-VM and actors

Sensor Framework

Sensors are deployed on every layer of the virtualized computer system. As shown in Figure 6.14, there are sensors in the hypervisor, guest userland, guest kernel and application containers. Apart from getting as much information as possible, this also enables one to compare and correlate information coming from different layers. For example, if an intruder is spoofing specific information for userland processes, one can compare this information with information coming from the guest kernel or the hypervisor. Depending on their location, the sensors differ in their implementation details, but all of them are designed to be minimally invasive in terms of resource consumption and performance penalties. Sensors are deployed without any extra analyzer functionality to keep the performance impact and additional attack surface introduced through them as minimal as possible. Consequently, communication with the Analysis-VM is crucial. Every sensor has two communication channels to interact with the Analysis-VM: one channel for the transmission of sensor data and the other channel for control messages. Control messages are used to manage sensors and rely on a custom wire protocol for control messages. In addition, the control channel is used to reconfigure a sensor if necessary (e.g., to reduce the amount of data being sent).

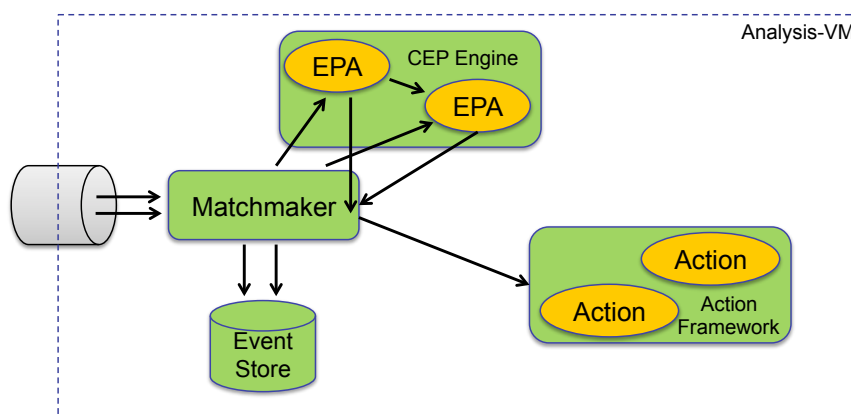


Figure 6.15: Analysis-VM

Analysis-VM

The Analysis-VM is the central instance of the monitoring system where sensor data is analyzed using CEP technology. This implies that the Analysis-VM is the main target of potential attackers. A successful breach could be used to compromise anomaly detection and sent harmful actions to actors, e.g., to shutdown sensors and/or VMs. For this reason, several approaches to improve security and consistency of the system have been combined to harden the Analysis-VM. Inside the Analysis-VM, Grsecurity⁴⁷, AppArmor⁴⁸ and Aide⁴⁹ are combined to get PaX address space protection⁵⁰, role-based access control, file system integrity checks, kernel auditing and executable protection. In addition, the Snort intrusion detection system is used to detect malicious network traffic. Furthermore, the host is configured to use SELinux⁵¹ protection for QEMU/KVM⁵² processes.

Figure 6.15 illustrates the different components responsible for event processing and triggering of actions. All events produced by the sensors are managed by a central hub called the *Matchmaker* that forwards them to the *CEP Engine* and to the *Event Store*. The latter is used as a historical database to store all events gathered from all sensors and can be used for what-if analysis or for machine learning to produce new rules. Since the number of events per second highly depends on the number of monitored VMs and number of deployed sensors, high throughput optimization is mandatory for this component. Live analysis is performed in the CEP Engine where different Event Processing Agents (EPAs) run continuous queries on the event streams to monitor the incoming sensor data. Queries are formulated in a simple but powerful language. The reaction time of the entire system depends on the runtime of the used CEP engine and how fast and how many EPAs it can handle. These EPAs can then trigger different *Actions* in the *Action Framework* that decides whether an action should run out-of-VM or should be submitted to any specific VM for execution.

Action Framework

Actors are deployed on all layers of the virtualized computer system (see Figure 6.14) to execute actions. They share the minimally invasive design principle and communication capabilities, but differ in their function. In contrast to sensors, actors receive cryptographically signed actions. Thus, they provide an execution environment and the capability to check signatures for their validity. The actions should be as flexible as possible to enable administrators to model any countermeasure required for a particular security intrusion.

6.4.4 Example Anomaly Detection

The general lifecycle of the monitoring system is depicted in Figure 6.16. In this figure the blue loop represents the sense-detect-react-cycle and the yellow loop shows the automatic rule generation process. To illustrate this new approach an example is given

⁴⁷<https://grsecurity.net/>

⁴⁸<http://apparmor.net>

⁴⁹<http://aide.sourceforge.net/>

⁵⁰<http://pax.grsecurity.net>

⁵¹<http://selinuxproject.org/>

⁵²<http://www.linux-kvm.org/>

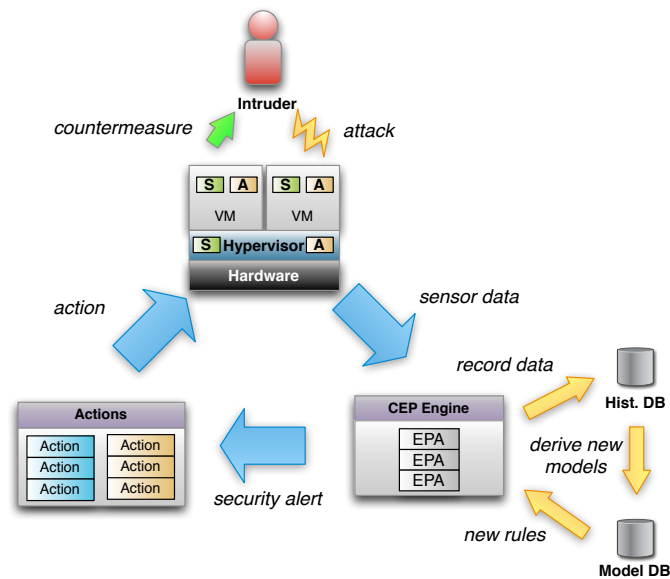


Figure 6.16: Monitoring lifecycle

in this subsection. An anomaly in double-entry accounting of the hypervisor and operating system layers port list is associated with a network based backdoor.

TCP Backdoor

In this scenario, an attacker has successfully installed a backdoor in a monitored virtual machine. He/she hides his/her presence through a rootkit, a modification of the operating system and its userland interfaces. Even though the backdoor is listening on an arbitrary TCP port, the process belonging to it and the listening socket are not listed by the operating system userland tools.

Sensors The scenario including the sensors and corresponding actions is shown in Figure 6.17. To detect the backdoor, at least two different sensors are involved: One sensor is running within the virtual machine and utilizes standard tools such as *netstat* to check for any listening sockets. Since the backdoor is well hidden, this sensor will not report the security breach.

The other sensor is inspecting the network state of the virtual machine from the hypervisor level. Since this sensor is running outside of the guest operating system, it is not affected by the backdoor's hiding features. On this level, an event is generated for the detection of a newly opened port in the virtual machine.

Analysis With the help of the Model Database and the Historical Database, queries can be generated to recognize normal or regular behavior. Therefore, an alarm should be triggered when a new open port is detected. Furthermore, by comparing both listening socket sensors, inside and outside of the virtual machine, it can be concluded that this really is a security related anomaly. A regular service installed in the virtual machine should not be hidden within the system. The conflicting sensors information is a clear sign of an attack.

6.4 Reactive Realtime Cloud Infrastructure Monitoring

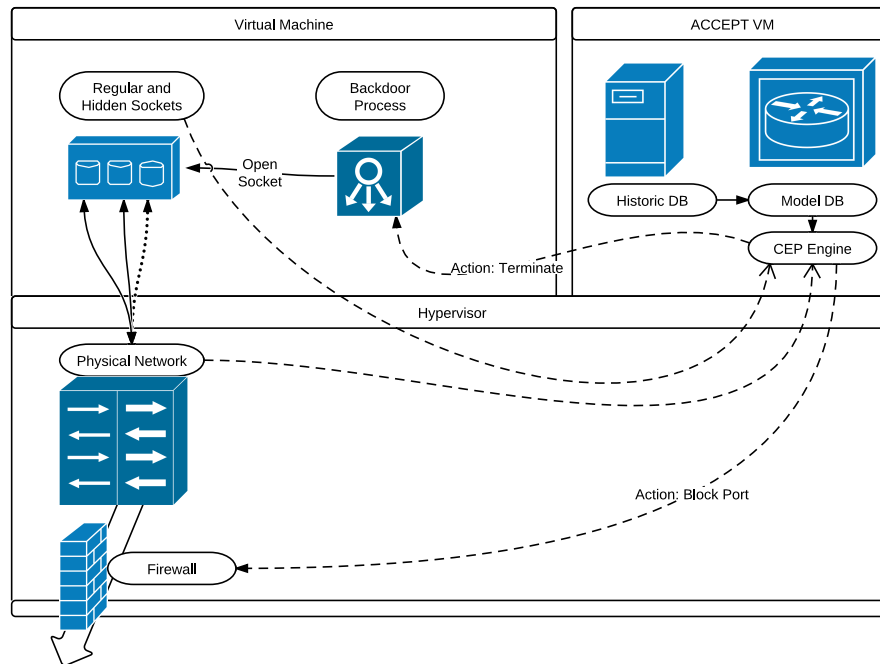


Figure 6.17: Example: TCP backdoor detection.

Action As a result of this attack, actions should be taken to eliminate the threat as much as possible. One such action could be to block all communication from and to the backdoor's port on the hypervisor level. This prevents the attacker of extracting information or further using the infected machine. Another step that should be taken is to isolate and possibly terminate the processes involved in the infection. For forensics purposes, taking a snapshot of the virtual machine and generating a dump is another possibility.

6.4.5 Sensor Framework

In contrast to existing monitoring frameworks, the reactive security monitoring system in this work incorporates sensors from different layers of a virtualized computer system with their own environments. Thus, sensors have to be written in different languages using specific libraries and programming idioms. An overview of the implemented sensors is shown in Table 6.6. To ensure that every sensor has a minimal memory footprint, is minimally invasive in terms of performance and can be implemented fast without having to know details about the Analysis-VM and the control messages, dedicated libraries and a framework for sensor execution were created. These libraries are accessible from C/C++, Python and Java to be useful on as many different layers as possible.

To ensure that the communication channel for event transmission between the sensors and the Analysis-VM offers maximum throughput and robust transfer, we considered different protocols and serialization formats.

JMS and AMQP are message broker protocols that require a dedicated broker and offer point-to-point and routed communication. While broker-oriented protocols have beneficial attributes such as guaranteed message transmission through persistent queues

Table 6.6: Number of implemented sensors

Layer	Quantity
Hypervisor	12
Guest Kernel	8
Guest Userland	39
Application Container	14

and easy bootstrapping, they are not optimized for high throughput. No parameter configuration leading to a sufficient throughput required was found.

In contrast to broker-oriented protocols, socket-oriented protocols offer higher throughput. UDP offers a very high throughput at the cost of possible packet loss, while TCP provides high throughput with reliable transmission. Since every event can contain information that might signal a security anomaly, TCP has been selected as the communication protocol for event transmission.

Human readable serialization formats such as XML and JSON, as well as binary marshalling algorithms such as MessagePack⁵³ and Google Protocol Buffers⁵⁴ were considered. While human readability in general creates an environment that is easier to debug, it enlarges the attack surface and has an impact on performance. Google Protocol Buffers offer smaller messages with faster transition than human-readable formats, but require a definition in the Interface Description Language, which makes them a suboptimal solution for dynamic adoption of sensors. MessagePack is also fast, but more flexible and dynamic, and thus the serialization format of choice. In addition, MessagePack offers libraries for different languages and was easier to port to the Linux kernel.

Furthermore, attributes related to the activity of sensors can be configured. In general, it is distinguished between sensors with a polling-based sending scheme and event driven sensors. While the latter obviously only send data when a specific event occurs, the sending rate of polling-based sensors can be configured. Thus, instead of relying on operating system log files, data is directly queried and watched for specific events to occur. In addition, the initial configuration of sensors including specific white- and blacklists can be altered within our framework.

Application Container

In today's production systems, enterprise applications run on application servers like Glassfish or Tomcat. To be able to get detailed information about the servers and the applications running on them, an environment for sensors written in Java was developed for this research. A dispatch sensor gets deployed and waits for directions delivered in the form of actions from the Analysis-VM. These actions contain names of sensors to be deployed and the Process Identification (PID) of a Java process to attach them to. Therefore, one can directly monitor the JVM of a given process. Using btrace⁵⁵, one is also able to trace internals of a process, such as class loading, JDBC operations,

⁵³<http://msgpack.org/>

⁵⁴<https://github.com/google/protobuf/>

⁵⁵<https://kenai.com/projects/btrace>

exceptions, webservices, file handling or URLs.

Userland

The userland is very broad, and thus there are many possible requirements for sensors. For this reason, the scripting language Python was adopted with its inherent prototyping qualities and several third-party packages. Moreover, a sensor developed in Python cannot easily compromise the security of the system since it is immune to memory corruption attacks such as buffer overflows or double frees. Furthermore, sensors within the framework are signed and only executed with a valid signature. This prevents malicious changes or the deployment of untrusted sensors.

As mentioned above, it is distinguished between event-driven sensors like a syslog sensor or a sensor that monitors specific files and folders, and polling-based sensors like a sensor for firewall rules or open network connections. Sensors can be full Python programs that make use of third-party packages or shell commands that are wrapped within the environment, making the creation of sensors an easy task without limitations in functionality. Using our Python sensor framework, a new sensor can be developed by writing less than 10 lines of code, which makes writing sensors for custom inhouse application monitoring a simple task.

Kernel

To create a communication channel to the Analysis-VM that uses the protocol (TCP) and serialization format (MessagePack) chosen, a kernel module that uses the Linux kernel's own socket library and a port of the MessagePack serializer to run in kernel space had to be written.

Sensors in this environment are written in C and trace specific system calls. For various reasons, there is no interface to trace system calls within the kernel. Since nearly every Linux distribution offers debug symbols for their kernels, the *sys_call_table* address can be determined using the *kallsyms* header. The addresses within this table can be replaced by the address of a decorator function that triggers an event before redirecting the call to the original system call. Since specific regions of the kernel are protected, this would trigger a general protection fault that can be suppressed temporarily.

Hypervisor

Hypervisor sensors can be grouped into sensors that leverage the libraries *libvirt*⁵⁶ and *libvmi*⁵⁷ and sensors that monitor the host operating system and QEMU/KVM processes. Since these libraries offer Python bindings, and monitoring the host operating system and specific processes is comparable to userland sensors, it was decided to use the Python sensor framework. Using high-level libraries for VM administration and introspection offers support for different hypervisors such as Xen, QEMU and KVM while keeping the code for the actual sensors small. In this research QEMU/KVM was used as hypervisor, because in QEMU/KVM virtual machines are processes that can be easily monitored, and the QEMU Monitor Protocol (QMP) can be leveraged to query

⁵⁶<http://libvirt.org/>

⁵⁷<http://libvmi.com/>

Table 6.7: Hardware specification of the host systems

Host	
CPU	Intel Core i7-4771 CPU @ 3.50GHz
RAM	32 GB
NIC	Intel I217-LM Gigabit LAN
HDD	LVM Raid 1 Seagate Barracuda ST3000
OS	Ubuntu 14.04 LTS
Kernel	3.13.0.39-generic

Table 6.8: Hardware specification of the guest systems

	Guest VM	Analysis-VM
RAM	4 GB	8 GB
Kernel	3.13.0.24	3.13.0-44
CPU	2 Cores	
NIC	VirtIO macvtap VEPA	
HDD	VirtIO	
OS	Ubuntu 14.04 LTS	

information regarding virtual hardware. Furthermore, the QMP can be used within actions to dump memory or alter a VM.

Experimental Evaluation

In this subsection, the environment is described in which the system was evaluated and the experiments that were performed to demonstrate that the system holds up to the claims made above. The evaluation platform uses a Gigabit network on multiple hosts with similar hardware specifications as shown in Table 6.7.

To produce realistic measurements, the guest system with the Analysis-VM was separated from the monitored guest system, both with similar specifications as listed in Table 6.8, and placed them on separate hosts.

Sensor Resource Usage To measure sensor resource usage, 28 of the userland polling-based sensors were configured to send events without any limitation. These sensors are encapsulated in a single process with one thread per sensor. During runtime, this process has been monitored using *ps* for 10 minutes. The results of this experiment are based on calculating the mean values of the percentual usage of CPU and RAM. The results listed in Figure 6.18 show the mean value of the percentual usage as the abscissa and the metric (CPU and RAM) as the ordinate. With values ranging from 1.9 to 2.6 percent leading to 2.2 percent mean CPU usage and 0.4 to 0.5 percent leading to 0.49 percent mean RAM usage, sensor resource usage is quite low, especially when considering the fact that the sensors had no limitations in their sending behavior.

Sensor Impact on Other Processes In this experiment, it was investigated how the activity of sensors affects other processes running under top load. For this purpose,

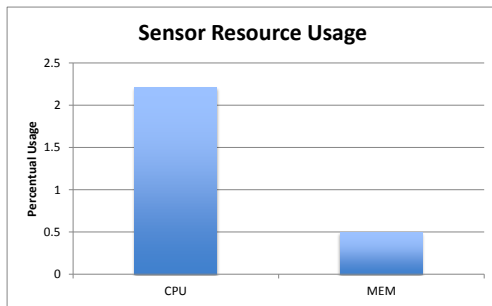


Figure 6.18: Resource usage of sensors

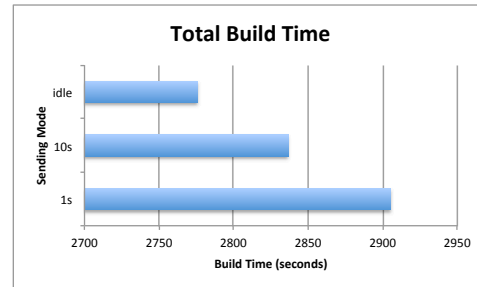


Figure 6.19: Comparison of sensor activity affecting total build time

the build time of Octave⁵⁸ 3.8.2 was measured without any sensor activity and two different sending modes with the Posix tool *time*. The results in Figure 6.19 show all three measurements with the build time in seconds as the ordinate and the different sensor setups grouped by user, system and idle time as the abscissa. There is nearly no impact between the different times resulting in a minimum total time of 2,776.02 seconds when no sensor is active and a maximum total time of 2,905.15 seconds when sensors send data every second, yielding a negligible delay of 4.65 percent.

6.4.6 Analysis-VM

This subsection presents the main components of the Analysis-VM, consisting of a federated CEP engine and a fast Event Store.

Federated CEP Engine

The analysis of the event streams coming from the sensors is performed in real-time using CEP technology. The use of CEP distinguishes the system from other security monitoring systems that use tailor-made event correlation engines. CEP engines usually provide powerful, declarative query languages, while tailor-made event correlation engines must often be configured via dedicated GUIs (e.g., web forms), configuration files or even imperative code. Using a declarative query language, complex queries are easier to express and, most importantly, allow for query optimization. Furthermore, CEP query languages not only allow developers to compose multiple queries to workflows, but also to express powerful queries, such as pattern matching queries, over event sequences (including correlations among the events of a pattern). However, tailor-made event correlation engines typically offer very good performance characteristics. The hypothesis is that general-purpose CEP engines can also provide very good performance in this specific application domain. In order to shed light on this hypothesis, the performance of the security monitoring system implemented with CEP technology inside was investigated.

In the Analysis-VM, not a single CEP engine is used alone, but a federation of heterogeneous CEP engines to exploit synergies between them with respect to functionality, expressiveness, and performance. Therefore, a federation of four different CEP engines allows one to execute a query or part of a query in the CEP engine that is most

⁵⁸<https://www.gnu.org/software/octave/>

suitable or performs best. In general, the federated CEP engine executes a query with a significantly better performance than a single CEP engine.

A serious issue with today's CEP engines is that there are absolutely no standards. Each individual CEP engine has its own query language in terms of syntax and, more critically, in terms of semantics. Since in this work a federated CEP engine is used that incorporates multiple heterogeneous CEP engines, an own query language had to be developed on top. All detection rules of the system are specified as queries in this query language that is based on SQL; Listing 6.1 gives an impression of the developed query language. Then, a query compiler translates every query into an abstract operator graph (i.e., an event processing network (EPN)). Each operator is identical to a basic event processing agent (EPA) that is supported. Currently, EPAs for event filtering, event aggregation, event correlation and event pattern matching are supported. For execution, another compiler translates each single EPA into the different native query languages of the CEP engines within the federation, while the query semantics are preserved. This approach is similar to application virtual machines such as the JVM or the CLR. Here, a program is compiled into platform-independent byte code. To execute a program on a specific platform, another compilation process translates the byte code into native code specifically for the given platform.

Before the abstract operator plan of a query is decomposed into sub-plans that are then distributed across the CEP engines of the federation, it is transformed by a query optimizer into a semantically equivalent but more efficient abstract operator plan. With respect to the supported types of EPAs, the rule-based query optimizer adopts well-known and powerful techniques from query compilers of database systems. In particular, filter EPAs are pushed down as close as possible to the event sources and a sub-graph consisting of multiple correlation EPAs (i.e., a multiway-correlation) is exchanged for a sub-graph containing the correlation EPAs in an optimal ordering according to a cost model.

Federation Manager After a new query has been successfully rewritten by the query optimizer, its abstract operator plan must be mapped to the federation of CEP engines. Conceptually, there are three different and concurrent optimization targets that determine an optimal mapping. First, the abstract operator plan should be distributed in a way such that the load is balanced. Otherwise, a CEP engine of the federation might get overloaded and become the bottleneck of the federation. Second, disjoint sub-plans of the abstract operator plan should be created and deployed so that every single EPA is executed by the best suited CEP engine of the federation from a performance point of view. Otherwise, an EPA of the query might be executed sub-optimally and become a bottleneck of the query. Third, sub-plans should be as large as possible. Otherwise, the communication overhead between the CEP engines becomes too high and decreases the overall performance of the system.

In the proposed system, a so-called federation manager automatically decomposes abstract operator plans and deploys the resulting sub-plans to different CEP engines. Of course, it respects the three concurrent optimization targets and tries to always find a good compromise between them. Since load balancing is a problem that has been solved in different environments, existing techniques were adopted to solve it in this context in an appropriate manner. However, the distribution of an operator plan across a federation of heterogeneous CEP engines is a novel problem requiring an adequate

solution. First, the individual strengths and weaknesses of each CEP engine of the federation had to be figured out. Therefore, comprehensive benchmarks (more than 200 individual benchmarks in total) had to be done to reveal this information. Each individual benchmark executed a certain type of EPA having a specific configuration of its parameters in all available CEP engines. For many of the benchmarks, the different CEP engines performed quite differently. Then the results of all benchmarks were manually evaluated and a classifier in the form of a decision tree was developed. This classifier gets an EPA and its parameter configuration as its input and returns the CEP engine that is expected to perform best. The federation manager uses the classifier to assign each EPA of an abstract operator plan individually to a CEP engine. Whenever multiple EPAs of the abstract operator plan that are directly connected with each other are assigned to the same CEP engine, they are clustered into a sub-plan. Finally, the resulting sub-plans can be deployed to their assigned CEP engines. Each CEP engine then executes autonomously all of its sub-plans. The federation manager as the master CEP engine is in charge of forwarding every incoming event to all CEP engines of the federation that execute at least one sub-plan that needs the event. All communication between the CEP engines of the federation (i.e., sending the output event of one sub-plan to a following sup-plan being executed in another CEP engine) is also done via the federation manager.

Query Index One of the CEP engines of the federation has been developed newly for this research and only supports event filtering (i.e., filter EPAs). Since the monitoring system must be able to execute workloads consisting of thousands of queries, an index for the queries is necessary in order to achieve sufficiently high performance. Unfortunately, most CEP engines do not include a query index, and the CEP engines that have a query index have unsatisfactory performance with respect to our target workload. Therefore, an own query index in the form of an additional CEP engine is provided.

The developed query index called BE^+ -tree is based on the BE-tree [253], the state-of-the-art query index. However, the BE-tree is a general-purpose query index that can be used in a variety of application domains including e-commerce, publish/subscribe and approximate string matching. The BE-tree achieves this wide applicability by supporting both fast index lookups and fast index updates (i.e., insertion of new queries) at the same time. In particular, the BE-tree is a dynamic tree-based index structure that handles insertions of new queries at runtime by local reorganization that is limited to only one path of the tree data structure. However, in the targeted applications, insertions of new queries occur with very low frequency compared to the frequency of new events.

Therefore, a new functionality was added to the BE-tree that creates a tree from scratch for a given query set. This gives two important advantages. First, a priori knowledge about the queries can be exploited to build better trees in comparison to the one-by-one design of the original BE-tree. The many optimizations performed by the BE^+ -tree are beyond the scope of this work. Second, the overall creation time of the BE^+ -tree is substantially lower than for the BE-tree. Due to the fast creation time of the BE^+ -tree, one can build an entirely new BE^+ -tree on updates even for large query sets very fast.

Experimental Evaluation The Analysis-VM is supposed to support the execution of huge query workloads with high performance. Due to the used query index, the federated CEP engine scales well with the total number of running queries, and due to the federation manager that optimizes the performance across a federation of heterogeneous CEP engines, every single query is executed with high event throughput and low latency. The following experimental evaluation indicates that the federated CEP engine in conjunction with a query index can master the challenging workloads of the security monitoring system.

```
SELECT *
FROM (SELECT COUNT(*) AS newCons
      FROM OpenConnections WINDOW(TIME 5 SECONDS)
      WHERE port=x AND vm=y)
MATCH RECOGNIZE (
  PATTERN abc
  WITHIN 10 SECONDS
  DEFINE a AS newCons > z * 1.5
         b AS newCons > a.newCons
         c AS newCons > b.newCons
)
```

Listing 6.1: Parametrized query of our workload

Listing 6.1 shows a typical anomaly detection query with parameters x , y and z of the query workload. It is used to detect attacks against services running on a server. The stream `OpenConnections` contains one event for each open connection to a VM on the server. Within each VM, a sensor periodically obtains all currently open connections and pushes one event for each open connection into `OpenConnections`. Each event comprises the name of the service, the identification of its process, the user it belongs to, the protocol of the connection, the identification of the VM (`vm`) on that the service is running, the port on which the service is running (`port`) and the IP address of the remote destination of the connection. The presented query is specifically for a service on port x at VM y . For the selected service, it counts the open connections within a sliding time window of size five seconds. On the output stream of the aggregation, the query searches for a pattern that indicates an abnormal use of the service (e.g., denial of service attack, brute force attack). However, the meaning of *normal* differs from service to service and, thus, is individual for every single service. The listed query uses the popularity of a service as its normal behavior z . Popularity is simply measured in the usual form of the total number of new connections within a time window of size five seconds. Obviously, different services have different popularities (e.g., an SSH server has significantly less new connections within a fixed time frame than a web server running a web site). Therefore, there is exactly one query running for each active service. The corresponding normal value z for an active service (x, y) can be easily determined by analyzing its history recorded by the Event Store (see the next subsection for details).

To evaluate the performance of the federated CEP engine, the number of active services was varied so that one could arbitrarily increase the total number of running queries. The events that the sensors emitted matched a running query with a probability of one percent. Figure 6.20 shows the overall performance of the federated CEP engine for 10,000 up to 80,000 monitored services and, thus, running queries. Note that each single query consisted of three EPAs in total (one filter EPA, one aggregation EPA and one pattern matching EPA) so that the total number of running EPAs is three

6.4 Reactive Realtime Cloud Infrastructure Monitoring

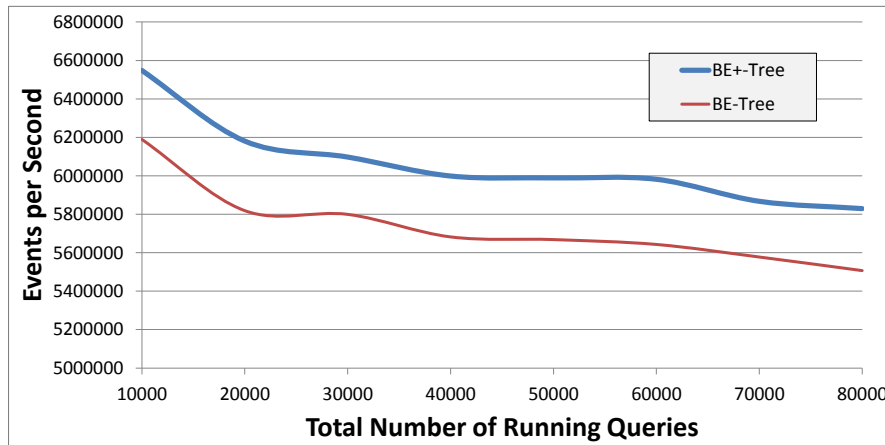


Figure 6.20: Performance of the federated CEP engine with a query index

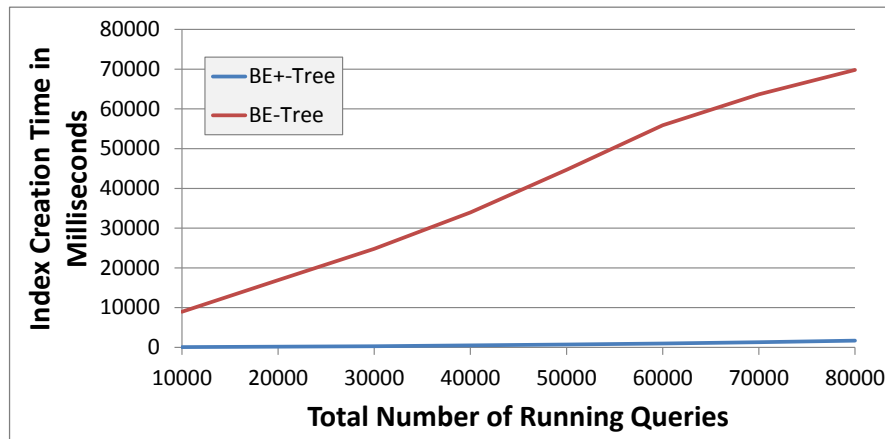


Figure 6.21: Query index creation time

times as much as the number of running queries. The figure shows that the federated CEP engine can handle more than 500,000 events per second in every case. Thanks to the query index, the performance scaled well with the number of running queries. If the number of running queries is increased, the performance decreases only slightly. Despite the fact that the filter expressions of the queries were quite simple, the BE⁺-tree still performed notably better than the BE-tree. For more complex queries, the performance improvements of the BE⁺-tree compared to the original BE-tree can be multiple factors. Figure 6.21 shows the creation times needed to create the query indexes of the last experiment from scratch. The graphs illustrate the better creation performance of the bulk loading implemented by the BE⁺-tree in comparison to the query-by-query loading of the BE-tree. Even the creation of an index for huge query workloads of size 80,000 required only about one second in case of the BE⁺-tree, while the creation of the BE-tree needed more than one minute. Due to its fast recreation on occasional updates of the query workload and its better overall performance, the BE⁺-tree is preferred over the BE-tree in the Analysis-VM.

Event Store

Apart from the real-time analysis of events, the Analysis-VM also supports offline analysis of historical data. Therefore, it is necessary to store all incoming events for a certain period of time (i.e., long-term storage). Due to the high data rates, existing storage solutions do not meet the requirements that are imposed on a data store for CEP systems. Traditional relational databases incur high transactional overhead, resulting in poor insertion performance. Key-value stores such as Cassandra⁵⁹ typically cannot keep up with the high data rates of CEP systems. The historical data store called *Event Store* is optimized for storing event stream data on a single machine, since it resides in the hardened Analysis-VM.

Design Principles The goal of the Event Store is to keep up with the event rates of the system while offering a reasonable (offline) query performance. For security and monetary reasons, the decision was made to use local hard drives as the primary storage medium. Thus, the Event Store is designed for sequential writing, aimed at storing millions of events per second.

The most common queries that need to be supported are *time travel queries* and *temporal aggregation queries*. Time travel queries allow requests for specific points and ranges in time, e.g., *all ssh login attempts within the last hour*. Temporal aggregation queries give a comprehensive overview of the data, e.g., *the average number of ssh logins for each day of the week during the last three months*.

The fastest way to store data on a disk is logging, i.e., to store each incoming event consecutively on disk and hence leverage its sequential write performance. The major drawback of this approach is its obviously poor query performance due to the lack of indexes. The opposite extreme would be to index each attribute of each stream in a multi-dimensional index, resulting in very poor insertion rates but very good query performance. The aim is to combine both: while high write performance is the main focus, the best possible query performance should also be supported.

Main Components The Event Store can be logically separated into three components: *store*, *event queues* and *workers*. The store is the central component, handling stream registration and event insertion. It keeps one event queue per stream, required for decoupling event queues from workers. Each worker resides in a dedicated thread and is assigned to a physical storage device (i.e., a hard drive). A worker's task is to process events of one or multiple dedicated event stream queues.

Figure 6.22 shows an example of a logical processing graph for 7 streams, 3 workers and 2 disks. The distribution of streams to workers and the assignment of workers to disks depends on the characteristics of the registered streams. The main objectives are (a) to optimally parallelize serialization and compression (worker distribution) and (b) to fully utilize disk speed (disk assignment).

Storage Layout Based on the awareness of the importance of the time dimension for query execution, it was figured out that the problem of storing incoming events in an indexed fashion on disks can be solved by an efficient bulk-loading strategy for B⁺-trees. The key attribute for the index is represented by the timestamps of events. Since events

⁵⁹<http://cassandra.apache.org/>

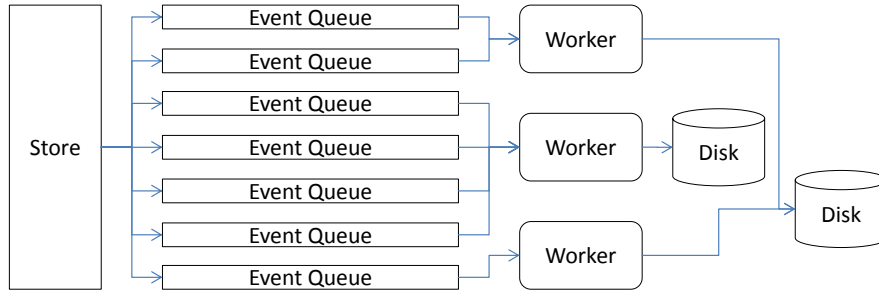


Figure 6.22: Example of an Event Store topology

arrive from the Analysis-VM in temporal order, a sort-based build approach was used. It is also possible to tolerate out-of order arrival within a certain tolerance, but this would require an explicit sorting step before writing the events into the Event Store.

For primary indexing, an augmented B^+ -tree is used. It is built in a bottom-up manner and avoids the traversal of the tree's right flank for each event. When a leaf node is filled, its corresponding index entry is inserted into the parent level. Therefore, one only has to access the parent node once per child node, which also applies to all higher levels of the tree. Due to the fact that one can hold the right flank in memory (only $\log_b N$ buffer blocks for N events and block size b needed), primary index creation is very fast.

To further improve write performance, the Event Store leverages data compression techniques. LZ4⁶⁰, an LZ-based, loss-less compression library, is used due to its fast compression speed. In terms of compression, the optimal physical storage layout would be a column layout. Unfortunately, a column layout leads to non-sequential I/Os, and thus causes a substantial performance loss. Thus, the storage layout is optimized for better compression rates using a hybrid approach [265]. The main idea is to store the relational events in a column-based fashion only within a single disk block.

Query Processing To speed up query performance, another characteristic of event data is utilized. As observed by Wang et al. [264], it can be assumed that values occurring within a small time interval are often very similar. This observation can be leveraged to speed up queries. For every node in the B^+ -tree, the minimum and maximum (\min_{a_i}, \max_{a_i}) value of each (selected) attribute a_i is stored for light-weight secondary indexing. Since the number of attributes i is negligible compared to the number of entries stored in a single node, the performance impact is minimal. During query processing, one can use these values for pruning. Additionally to the minimum and maximum values, each node stores the count of all underlying entries as well as the sum of each attribute, which allows it to answer aggregation queries very fast in logarithmic time.

For full secondary indexing, a write-optimized indexing technique is used, namely cache-oblivious look-ahead arrays (COLA) [266]. Since full secondary indexing is costly compared to the primary index, indexes are only created partially if the system load is sufficiently low.

⁶⁰<https://code.google.com/p/lz4/>

Experimental Evaluation To measure the write performance of the Event Store, two metrics are used: the number of events stored per second and the (gross) data rate while storing on disk. Additionally, the net data rate is used due to its physical limitation while writing on disk. Another limiting factor is CPU performance. As the Event Store has been implemented in Java, data serialization is a major performance issue. Furthermore, data compression also requires CPU time.

The goal is to leverage the maximum net data rate (i.e., disk rate) with even higher gross data rate (i.e., good compression ratios). For the experiments, events consisting of 10 attributes of 8 byte each (80 bytes per event) were used.

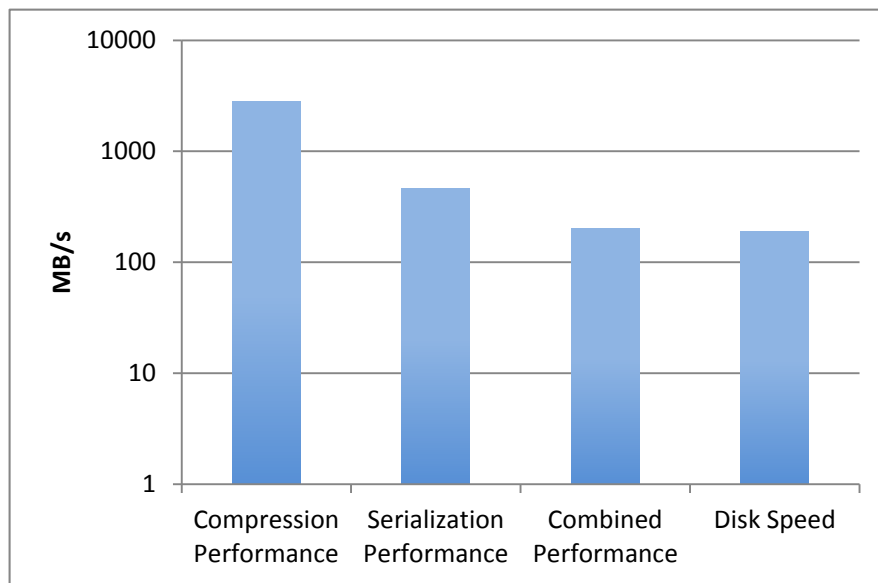


Figure 6.23: Evaluation of system performance factors

In the first experiment (see Figure 6.23; note the log scale), the key performance factors in the Event Store (and Java) were evaluated. The results indicate that the maximum possible gross data rate for 8 KB blocks with compression (but without serialization) is about 2818 MB/s, which is far beyond the disk speed of 187.4 MB/s. The main bottleneck of the Event Store is the serialization process, which is due to the Java type system. Serialization of event data results in 461 MB/s (net=gross) data rate. Overall, the total gross data rate of the Event Store is about 201 MB/s, resulting in a net data rate of 88 MB/s. Although the Event Store is CPU-bound for a single stream, its gross data rate already surpasses the sequential write speed of the disk (187.4 MB/s) and reaches 2.64 million events per second. Note that the net data rate is about 47 % of the available disk speed.

In the second experiment shown in Figure 6.24, the write performance of two (parallel) streams, on separate workers but on the same disk, were evaluated since the results of the first experiment suggest higher possible data rates. If one increases the number of streams and therefore the degree of parallelization, compression can improve the gross data rate significantly, reaching 186 % of the physical disk speed (4.56 million events per second). Even two streams do not suffice to utilize full disk speed, since the configuration is still CPU-bound.

The third experiment evaluated the maximum reachable gross data rate for more

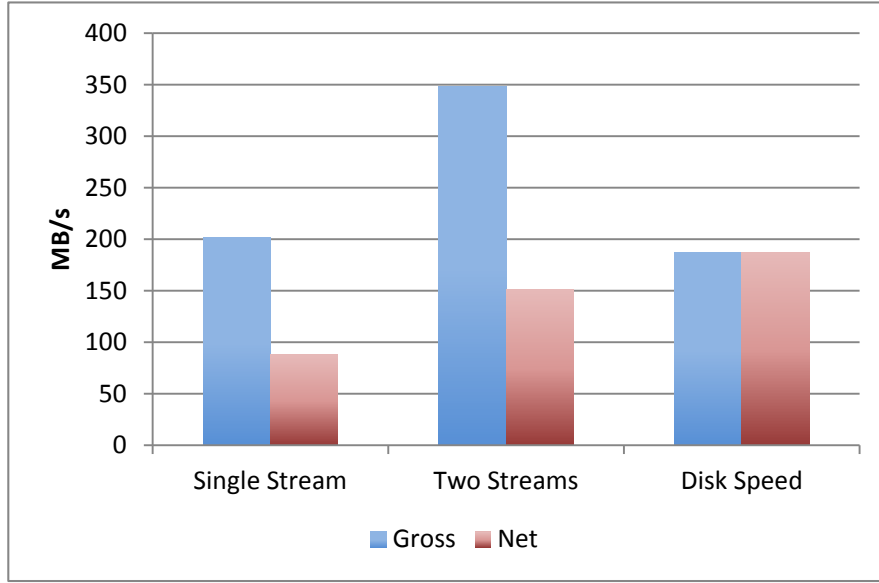


Figure 6.24: Comparison of net and gross data rates

than two parallel streams. The Analysis-VM was extended to 8 cores and processed each stream on a separate worker thread. In this scenario, it is assumed that at least 4 physical disks are installed. Since the test system offered only one disk, the final write command to physical storage was omitted. Figure 6.25 shows the measured number of events per second. With 8 parallel streams, a gross data rate of 768.6 MB/s and 9.05 million events per second is reached.

For query evaluation purposes, the relation scan performance (Q_1), the time travel performance (Q_2) as well as the temporal aggregation performance (Q_3) was measured on a data set with 20 million events (with 80 bytes per event). For the time travel and temporal aggregation performance, a continuous fraction containing 20 % of the data, i.e., 4 million events, was queried and its query time was measured.

Figure 6.26 shows the results of the query evaluation (note the log scale). An entire relation scan of the stream (Q_1) takes about 7.1 seconds, reaching a gross data rate of 213 MB/s. The time travel query (Q_2) using the index takes only about 1.3 seconds, less than a fifth of a relation scan. Finally, the temporal aggregation query (Q_3) of the same partition as in Q_2 only takes 5 milliseconds.

6.4.7 Action Framework

The proposed reactive security monitoring system can not only observe security intrusions and malicious behavior, but also trigger countermeasures to prevent an attack or further system breaches. For this purpose, an action framework, as described below, was developed.

Actions and Actors

The action framework has been developed to perform actions, when the matchmaker for EPAs triggers events. Since possible countermeasures should be both simple and

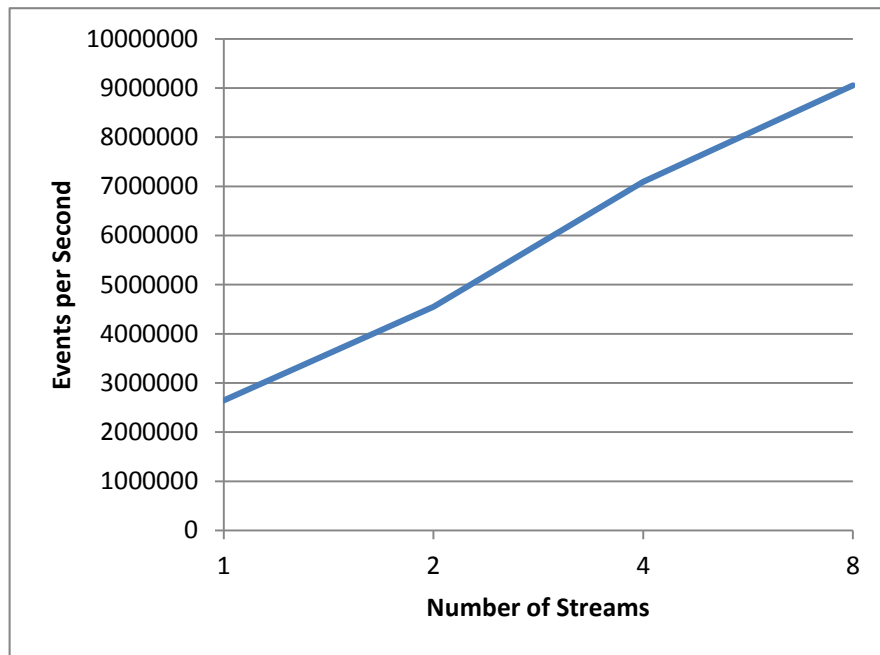


Figure 6.25: Increasing the number of streams

flexible, the decision was made to use Python that due to CPython and Jython⁶¹ could be easily integrated in the environment. An action consists of a signed Python script and an XML file that contains its required parameters and the layer on which the action should be executed. Similar to sensors, actions can be dynamically added and removed and can be executed on all layers of the virtualized computer system. In addition, every action sent to an actor for execution can be configured to use the communication channel used in the system. In this way, feedback whether or not the operation was successful is provided, or another action based on the results of its execution can be triggered. Typical actions are: making memory dumps with a forensic analysis and performing live migration (hypervisor level), creating firewall rules and killing processes (userland level), and altering JVM settings including restrictions (application container level). Actors executing actions are only in charge of registering themselves, check the signature of incoming actions and provide an execution environment including parameters relevant for the specific layer.

Experimental Evaluation

```
SELECT *
FROM TestStream
WHERE identifier = "BenchmarkSensor"
```

Listing 6.2: Query used for benchmarking

To demonstrate that the proposed reactive security monitoring system can be used in a real world setting, the difference between the timestamp of the creation of an event at a sensor and the timestamp at the execution of an action triggered due to that

⁶¹<http://www.jython.org/>

6.4 Reactive Realtime Cloud Infrastructure Monitoring

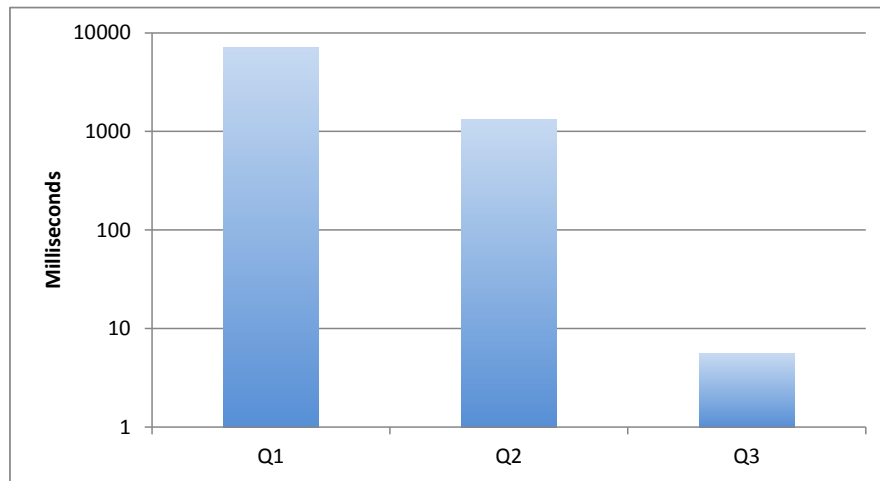


Figure 6.26: Query performance evaluation (overall query time) for queries Q1, Q2 and Q3

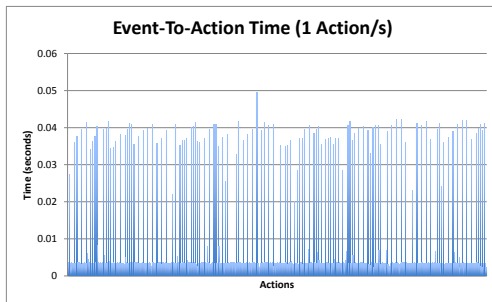


Figure 6.27: Time from event to action (single)

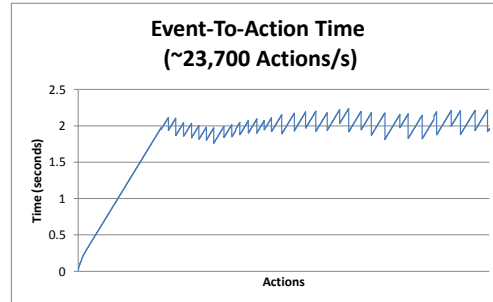


Figure 6.28: Time from event to action (bulk)

event was measured. A sensor explicitly designed for this benchmark sends an event containing an increasing identifier and the current timestamp of the guest machine. The experiment is based on a simple query shown in Listing 6.2 to redirect this event to the action framework. An action that prints the current timestamp and expects this ID is used and sent to an userland actor where the timestamps are then compared, resulting in the complete event-to-action time for this workflow. The actor is placed on the same guest machine to avoid having to synchronize different clocks. Figure 6.27 shows the measured results for an event-to-action time of one event and thus one action per second with a time series as the ordinate and the difference in timestamps as the abscissa. The results indicate that the system has an average reaction time of 0.007 seconds.

Figure 6.28 shows the measurement series for an event-to-action time with no limitation in the sending rate, resulting in around 23,700 actions per second in which the time from event to action is measured. Similar to the previous experiment, the figure shows a time series as the ordinate and the difference in timestamps as the abscissa. Since this experiment investigates a system at the boundaries of the virtual hardware with a triggering rate of over 23,000 actions per second, a mean value of 1.82 seconds for the time from event creation to the resulting action is still within an acceptable timeframe and indicates how well the system operates under heavy load.

6.4.8 Conclusion

In this section, a novel approach for reactive security monitoring in a virtualized computer environment based on minimally-intrusive dynamic sensor deployment vertically across virtualization layers and horizontally within a virtual machine was presented. The sensor streams are analyzed using a federation of CEP engines and a query index to maximize the performance queries, and the results of the analysis are used to trigger appropriate actions in response to the detected security anomalies. Furthermore, a novel event store is utilized that supports fast event logging for offline analysis of collected historical data. Experiments have shown that the proposed system can execute tens of thousands of detection rules with high performance and low latency.

There are several areas of future work. For example, the research has demonstrated that the system can be used for reactive security monitoring with respect to performance, but not how it should be used. Thus, specifying appropriate sensors, CEP queries and actions for several types of potential attacks is a challenging task for future research. Furthermore, research on using machine learning techniques to automatically set parameters of CEP queries based on historical data collected in the Event Store is required. Finally, to increase the processing speed further and handle an even larger number of events, multi-core architectures should be leveraged and certain CEP queries should be offloaded to General Purpose Graphic Processing Units (GPGPUs).

6.5 Summary

In Section 6.2, an empirical study of the security properties of email server communication within the German IP address space range was presented. Instead of investigating end-user security or end-to-end encryption, we focused on the connections between SMTP servers relying on transport layer security. An in-depth analysis was performed on the involved ciphers suites, the certificates used and certificate authorities, and the behavior of email providers when communicating with improperly secured email servers. Conclusions drawn from this analysis lead to several recommendations for mitigating the security issues currently present in the email system as it is deployed in the Internet.

An approach for combined malware detection and kernel rootkit prevention in virtualized cloud computing environments was presented in Section 6.3. All running binaries in a virtual instance are intercepted and submitted to one or more anti-virus analysis engines. Besides a complete check against a signature database, live introspection of all system calls is performed to detect yet unknown exploits or malware. Furthermore, to prevent that an intruder retains persistent control over a running instance after a successful compromise, an in-kernel rootkit prevention approach is proposed. Only authorized and thus trusted kernel modules are allowed to be loaded during runtime; loading of unauthorized modules is no longer possible. Finally, the performance of the presented solutions was evaluated and showed its practicability.

Finally, in Section 6.4, a new approach for reactive security monitoring in a virtualized computing environment based on minimally-intrusive dynamic sensor deployment vertically across virtualization layers and horizontally within a virtual machine instance was presented. The sensor streams are analyzed using a novel federation of Complex Event Processing engines and an optimized query index to maximize the performance of continuous queries, and the results of the analysis are used to trigger appropriate actions on different virtualization layers in response to detected security anomalies. Furthermore, a novel event store that supports fast event logging is utilized for offline analysis of collected historical data. Experiments show that the proposed system can execute tens of thousands of complex detection rules and trigger actions efficiently and with low latency.

7 Conclusion

7.1 Summary

In this thesis, existing technology used during disaster scenarios was evaluated, and shortcomings were identified regarding usability, feature-completeness, and security. Several novel approaches were developed to improve the usefulness of a decentralized communication system suited specifically for challenged networks such as the ones found during an emergency. Overall, several improvements were made and novel technologies were designed for secure emergency communication in course of this thesis.

MiniWorld, a novel, flexible distributed emulation environment was developed. It enables us to evaluate software for emergency scenarios in a more realistic way. This is achieved by combining full system emulation with interchangeable network back-ends to simulate wireless properties such as the ones found with Bluetooth or WiFi links and allows an orchestrated dynamic movement of nodes.

Furthermore, the usefulness of DTN and Device-to-Device networks was improved by novel approaches and algorithms for local announcements and data dissemination. Also, a first of its kind RPC system specifically tailored to challenged network conditions in an emergency scenario, called *DTN-RPC*, was presented. These developments are the foundation for novel applications such as mobile optimized on-device face recognition with D2D data distribution capabilities (*SmartFace*) and an integrated approach utilizing unmanned ground and aerial vehicles (*uv4ec*). Through the addition of microcontroller units and making long range radio transceivers available to mobile device, the system can be used in various scenarios including environmental monitoring. Moreover, steps were taken to increase the security and robustness of the presented disruption-tolerant device-to-device emergency communication system.

To increase the security of mobile apps, novel tools for large scale dynamic (*Dynalyze*) and static (*AndroLyze*) analysis of Android apps were developed. The importance of such tools is shown by audits of common emergency communication apps as well as large-scale tests for the use of SSL/TLS and various cryptographic functions in apps.

Finally, a novel system for increased security and monitoring of virtualized servers systems was developed. The combination of integrated classic anti-virus systems with live introspection and dynamically deployed sensors across several layers enables new ways to defend a system, with minimal footprint on each guest machine. A federated CEP engine in conjunction with a historic event database helps identifying anomalies and reducing false alarms.

Overall, this thesis has presented several solutions to provide a secure and flexible communication system to cope with the challenging network conditions during a disaster scenario, ranging from DTN D2D communication on mobile devices utilizing secure communication channels to Internet connected virtualized cloud systems.

7.2 Future Work

There are several areas for future research regarding secure communication in emergency scenarios. Many points have already been highlighted in the individual conclusions of each section. Furthermore, future research may include general improvements and extensions of the solutions presented in this thesis, which will be discussed in the following.

Disruption-Tolerant Device-to-Device Emergency Communication

Although the solutions presented in this thesis can be used in various scenarios, there is room for further optimizations, especially for low-bandwidth links, such as LoRa. Here, DTN distribution strategies and protocols can be further developed to reduce the amount of bandwidth needed. Also, the resource footprint of security related functions, such as signing and hashing, is challenging for embedded microcontrollers. In terms of implementation, the solutions are engineered mainly towards Linux, macOS, and Android. However, they should also be expanded to iOS and Windows for wider real world adoption.

Security Vulnerability Analysis of Mobile Apps

While the dynamic and static security checks designed during the course of this thesis have already unveiled many vulnerabilities in existing emergency communication apps, some aspects should be investigated further. Automatic and intelligent combination of both analysis methods could be used to eliminate false positives, and integration of these security checks into software development tools could further improve mobile app security.

Secure Cloud Systems

The VM introspection and monitoring systems presented in this thesis are heavily tailored towards KVM and the Linux kernel. For wider adoption, including support for other hypervisors and operating systems such as FreeBSD or Windows Server would be helpful. Building upon this foundation and the amount of sensor data that can be acquired, the possibility of using machine learning techniques for anomaly detection should be investigated.

List of Figures

1.1	Natural catastrophes since 1980	1
1.2	FEMA Top 10 Response Core Capabilities	2
1.3	FEMA Top 10 Protection Core Capabilities	2
2.1	IP vs DTN data flow	10
2.2	The Serval technology stack	11
2.3	Contents of NINA APK	13
2.4	DWD WarnWetter app in action	15
2.5	BBK NINA in action	16
2.6	KATWARN in action	16
2.7	FireChat room overview	18
2.8	Serval Mesh main view	18
2.9	Cloud services overview	19
3.1	Typical Internet services used during disasters	23
3.2	Mobile apps deployed on different devices	24
3.3	Thesis problem overview	25
3.4	Secure communication during disaster scenarios	27
4.1	DT D2D communication system developments	32
4.2	<i>MiniWorld's</i> Architecture	35
4.3	<i>MiniWorld's</i> Distributed Architecture	38
4.4	<i>MiniWorld</i> Implementation	39
4.5	Boot Times: OpenWrtBB vs. Debian8 (<i>Shell Prompt</i>)	42
4.6	Boot Times: Snapshot Boot vs. Real Boot (Debian8)	42
4.7	Network Backend Throughput	43
4.8	RTTs for the Network Backends	43
4.9	Topology Switching (<i>Bridged WiFi</i>)	44
4.10	Distributed Mode: Boot Times	45
4.11	Distributed Mode: Differential Topology Switching	45
4.12	Distributed Mode: Tunnel Overhead	46
4.13	<i>MF Mixed</i> : Rhizome store size, network and CPU load	54
4.14	Mass-Messages CPU Usage over Time	55
4.15	<i>Hub</i> limited <i>PM</i> : Rhizome store size, network & CPU	56
4.16	Evaluation Scenario: Chained Limited Medium	57
4.17	Energy consumption of announcement intervals	58
4.18	Serval Power Consumption	59
4.19	Drive-by store-and-forward data exchange	60
4.20	Drive-by window of opportunity example	60
4.21	Announcements/second in a static network of 25 nodes	68
4.22	Announcement Strategies Comparison	69

List of Figures

4.23	Splitting network configuration with 10 nodes	71
4.24	A growing network with 100 nodes	72
4.25	<i>Announcement Gaps</i> in a static network of 10 nodes	73
4.26	Calling a remote procedure in a DTN disaster scenario	75
4.27	DTN-RPC flowchart for client and server.	79
4.28	DTN-RPC Bandwidth and CPU Usage	85
4.29	Round trip times in different topologies	86
4.30	DTN vs. Non-DTN for 100MB with Islands Topology	87
4.31	Base station for monitoring, relaying and processing	93
4.32	BLE LoRa modem with plain Raspberry Pi 3 for size comparison	95
4.33	Waterproof static sensor box deployed	98
4.34	Communication range of different LoRa setups	99
4.35	RSSI vs. Distance	100
4.36	Number of received packets in relation to packets sent.	100
4.37	Basic two-stage face detector	105
4.38	Two-stage face detector, preprocessing, and parameters	106
4.39	<i>SmartFace</i> in action	106
4.40	<i>SmartFace</i> implementation	107
4.41	Examples from image test set	108
4.42	Comparison of different color spaces and depths	108
4.43	Face-Detection Performance for Different Resolutions	109
4.44	Comparison of cropped areas from grey scale images	109
4.45	Overall benchmark	111
4.46	Best of all categories dlib vs <i>SmartFace</i>	111
4.47	Direct comparison of devices	111
4.48	Individual device performance	112
4.49	<i>SmartFace</i> Performance on Various Devices and Links	113
4.50	Emergency communication scenario	116
4.51	Architecture of <i>NICER OCC</i>	119
4.52	UI component of <i>NICER OCC</i>	120
4.53	Hector Tracker Robot in Simulated Disaster	121
4.54	Robot sensor data for a "victim found" event	122
4.55	UAV/UGV Simulation Setup	123
4.56	Opportunities for data exchange	124
4.57	Opportunities for data exchange over time	124
4.58	File distribution times	124
4.59	Message distribution times	124
4.60	SEMUD vs OLSR Reaction Times	125
4.61	Secure Key Management Illustration	130
4.62	Flooding attack and revocation	133
5.1	Insecure mobile AV solution compromised	148
5.2	Sample SSL Warning Message	151
5.3	Facebook's SSL warning	152
5.4	Analysis approach of <i>AndroLyze</i>	162
5.5	Typical workflow while using <i>AndroLyze</i>	164
5.6	Parallelization of APK import	168

5.7	Relation between script requirements and runtime	169
5.8	Local parallel mode +Source code	170
5.9	Comparison of job scheduling strategies	170
5.10	Local parallel mode +Bytecode	171
5.11	Distributed parallel mode + SSL vs. local parallel mode +SSL	171
5.12	APK distribution	172
5.13	Performance of AndroLyze	173
5.14	Crypto Statistics for Top Free 500 Archive	174
5.15	Dynalize platform architecture	177
5.16	Dynalize IaaS Layout	178
5.17	Screenshot of the Dynalize interface	180
5.18	Virtual device start time and throughput on EC2	182
5.19	Storage throughput for different EC2 instance types	183
5.20	Container startup	184
5.21	Average throughput and overhead of storage Backends	185
5.22	Successful MITM attack on WarnWetter	189
5.23	Successful MITM attack on NINA	190
5.24	Successful MITM attack on Sicher Reisen	192
5.25	Successful MITM attack on BIWAPP	193
5.26	Different successful attacks on FEMA app	195
5.27	Successful MITM attack on Disaster Alert	196
5.28	Successful MITM attack on NOAA Weather Radar	197
6.1	Email transfer and TLS usage.	202
6.2	SMTP and TLS usage among the scanned hosts	205
6.3	SSL and TLS versions used in the scanned services	206
6.4	Use of cipher suites	207
6.5	Change of the average key lengths over time.	210
6.6	Categorization of services using the main security properties	210
6.7	Root certificates in relation to their signed server certificates	210
6.8	CA Topology of the Comodo CA.	212
6.9	Malware scanner architecture	221
6.10	Authorized module loading state transition diagram	224
6.11	Module loading activity	227
6.12	Malware Detection Benchmarks	228
6.13	Module (un)loading overhead measurement	229
6.14	Sensors, Analysis-VM and actors	236
6.15	Analysis-VM	236
6.16	Monitoring lifecycle	238
6.17	Example: TCP backdoor detection.	239
6.18	Resource usage of sensors	243
6.19	Comparison of sensor activity affecting total build time	243
6.20	Performance of the federated CEP engine with a query index	247
6.21	Query index creation time	247
6.22	Example of an Event Store topology	249
6.23	Evaluation of system performance factors	250
6.24	Comparison of net and gross data rates	251

List of Figures

6.25	Increasing the number of streams	252
6.26	Query performance evaluation	253
6.27	Time from event to action (single)	253
6.28	Time from event to action (bulk)	253

List of Tables

4.1	Network Topologies	51
4.2	Scenario Tests	52
4.3	Test File Sets	52
4.4	Announcements of the strategies compared.	70
4.5	Correlation of energy consumption and announcements	73
4.6	Topologies	83
4.7	Overview of SBC and MCU Platforms	96
4.8	Overview of different Sensors	97
4.9	Power consumption of MCUs including radio transceivers	98
4.10	Comparison of Concept Detection on Pi vs. Neural Compute Stick	100
4.11	Cost of BLE LoRa Modem	101
4.12	Example configuration for a MSP	101
4.13	Example configuration for a SSP (w/o power supply)	102
4.14	Device Specifications	107
4.15	Contingency table for <i>dlib</i>	110
4.16	Contingency table for <i>OpenCV</i>	110
4.17	Contingency table for <i>SmartFace</i> (faster runtime)	110
4.18	Contingency table for <i>SmartFace</i> (higher quality)	110
4.19	Transmission times for various link types	112
4.20	Performance test of OCC w.r.t message processing	126
5.1	Top 10 hosts in all extracted URLs	144
5.2	Trust Managers & Socket Factories that trust all certificates	145
5.3	Results of the SSL pinning analysis	150
5.5	APK test sets	168
5.6	Script test sets	168
5.7	Prices of the used instance types	183
5.8	Emergency warning app audit results	198
6.1	Shares of weak and broken cipher suites	208
6.2	Time period of validity of the retrieved certificates	209
6.3	Self-signed, server and CA certificates	209
6.4	Top 10: Most popular CAs	211
6.5	Email provider strategies for connections to other email servers	213
6.6	Number of implemented sensors	240
6.7	Hardware specification of the host systems	242
6.8	Hardware specification of the guest systems	242

Bibliography

- [1] N. Schmidt, L. Baumgärtner, P. Lampe, K. Geihs, and B. Freisleben, “MiniWorld: Resource-aware Distributed Network Emulation via Full Virtualization,” in *22nd IEEE Symposium on Computers and Communication (ISCC 2017)*, Heraklion, Greece: IEEE, 2017, pp. 818–825 (cit. on pp. [xiii](#), [6](#), [33](#)).
- [2] L. Baumgärtner, P. Gardner-Stephen, P. Graubner, J. Lakeman, J. Höchst, P. Lampe, N. Schmidt, S. Schulz, A. Sterz, and B. Freisleben, “An Experimental Evaluation of Delay-Tolerant Networking with Serval,” in *IEEE Global Humanitarian Technology Conference (GHTC 2016)*, Seattle, USA: IEEE, 2016, pp. 1–8 (cit. on pp. [xiii](#), [7](#), [49](#), [62](#), [76](#), [91](#)).
- [3] L. Baumgärtner, P. Graubner, J. Höchst, A. Klein, and B. Freisleben, “The More You Speak, the Less You Hear: On Dynamic Announcement Intervals in Wireless On-demand Networks,” in *13th Conference on Wireless On-demand Network Systems and Services (WONS 2017)*, Jackson Hole, USA: IEEE, 2017, pp. 33–40 (cit. on pp. [xiii](#), [6](#), [61](#)).
- [4] A. Sterz, L. Baumgärtner, R. Mogk, M. Mezini, and B. Freisleben, “DTN-RPC: Remote Procedure Calls for Disruption-Tolerant Networking,” in *IFIP Networking 2017 Conference and Workshops (Networking 2017)*, Stockholm, Sweden: IFIP, 2017, pp. 1–9 (cit. on pp. [xiii](#), [6](#), [76](#)).
- [5] P. Lampe, L. Baumgärtner, R. Steinmetz, and B. Freisleben, “SmartFace: Efficient Face Detection on Smartphones for Wireless On-demand Emergency Networks,” in *24th International Conference on Telecommunications (ICT 2017)*, Limassol, Cyprus: IEEE, 2017, pp. 1–7 (cit. on pp. [xiii](#), [6](#), [92](#), [99](#), [103](#)).
- [6] L. Baumgärtner, A. Penning, P. Lampe, B. Richerzhagen, R. Steinmetz, and B. Freisleben, “Environmental Monitoring Using Low-Cost Hardware and Infrastructureless Wireless Communication,” in *IEEE Global Humanitarian Technology Conference (GHTC 2018)*, San Jose, USA: IEEE, 2018, accepted for publication (cit. on pp. [xiii](#), [6](#), [89](#)).
- [7] L. Baumgärtner, S. Kohlbrecher, J. Euler, T. Ritter, M. Schmittner, C. Meurisch, M. Mühlhäuser, M. Hollick, O. von Stryk, and B. Freisleben, “Emergency Communication in Challenged Environments via Unmanned Ground and Aerial Vehicles,” in *IEEE Global Humanitarian Technology Conference (GHTC 2017)*, San Jose, USA: IEEE, 2017, pp. 1–9 (cit. on pp. [xiii](#), [6](#), [91](#), [115](#)).
- [8] F. Kohnhäuser, M. Schmittner, L. Baumgärtner, L. Almon, S. Katzenbeisser, M. Hollick, and B. Freisleben, “SEDCOS: A Secure Device-to-Device Communication System for Disaster Scenarios,” in *42nd Annual IEEE Conference on Local Computer Networks (LCN 2017)*, Singapore, Singapore: IEEE, 2017, pp. 195–198 (cit. on pp. [xiii](#), [6](#), [128](#)).

Bibliography

- [9] S. Fahl, M. Harbach, T. Muders, L. Baumgärtner, B. Freisleben, and M. Smith, "Why Eve and Mallory Love Android: An Analysis of Android SSL (In)Security," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security (CCS)*, Raleigh, USA: ACM, 2012, pp. 50–61 (cit. on pp. [xiv](#), [7](#), [139](#), [159](#), [160](#)).
- [10] L. Baumgärtner, P. Graubner, N. Schmidt, and B. Freisleben, "AndroLyze: A Distributed Framework for Efficient Android App Analysis," in *IEEE 2nd International Conference on Mobile Services (MS 2015)*, New York City, USA: IEEE, 2015, pp. 73–80 (cit. on pp. [xiv](#), [7](#), [159](#)).
- [11] P. Graubner, L. Baumgärtner, P. Heckmann, M. Müller, and B. Freisleben, "Dy-nalize: Dynamic Analysis of Mobile Apps in a Platform-as-a-Service Cloud," in *IEEE 8th International Conference on Cloud Computing (CLOUD 2015)*, New York City, USA: IEEE, 2015, pp. 925–932 (cit. on pp. [xiv](#), [7](#), [175](#)).
- [12] L. Baumgärtner, J. Höchst, M. Leinweber, and B. Freisleben, "How to Misuse SMTP over TLS: A Study of the (In) Security of Email Server Communication," in *2015 IEEE Trustcom/BigDataSE/ISPA*, Helsinki, Finland: IEEE, 2015, pp. 287–294 (cit. on pp. [xiv](#), [7](#), [203](#)).
- [13] M. Schmidt, L. Baumgärtner, P. Graubner, D. Böck, and B. Freisleben, "Malware Detection and Kernel Rootkit Prevention in Cloud Computing Environments," in *19th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP 2011)*, Ayia Napa, Cyprus: IEEE, 2011, pp. 603–610 (cit. on pp. [xiv](#), [7](#), [217](#)).
- [14] L. Baumgärtner, P. Graubner, M. Leinweber, R. Schwarzkopf, M. Schmidt, B. Seeger, and B. Freisleben, "Mastering Security Anomalies in Virtualized Computing Environments via Complex Event Processing," in *Proceedings of the The Fourth International Conference on Information, Process, and Knowledge Management (eKNOW 2012)*, Valencia, Spain: IEEE, 2012, pp. 76–81 (cit. on pp. [xiv](#), [7](#), [232](#)).
- [15] L. Baumgärtner, C. Strack, B. Hoßbach, M. Seidemann, B. Seeger, and B. Freisleben, "Complex Event Processing for Reactive Security Monitoring in Virtualized Computer Systems," in *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems*, Oslo, Norway: ACM, 2015, pp. 22–33 (cit. on pp. [xiv](#), [7](#), [232](#)).
- [16] D. M. West and M. Orr, "Race, Gender, and Communications in Natural Disasters," *Policy Studies Journal*, vol. 35, no. 4, pp. 569–586, 2007 (cit. on p. [1](#)).
- [17] L. Comfort and T. Haase, "Communication, Coherence, and Collective Action: The Impact of Katrina on Communications Infrastructure," *Public Works Management & Policy*, vol. 10, no. 4, pp. 328–343, 2006 (cit. on p. [1](#)).
- [18] R. Mogk, L. Baumgärtner, G. Salvaneschi, B. Freisleben, and M. Mezini, "Fault-tolerant Distributed Reactive Programming," in *32nd European Conference on Object-Oriented Programming (ECOOP 2018)*, vol. 109, Amsterdam, The Netherlands: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2018, 1:1–1:26 (cit. on p. [6](#)).

- [19] P. Graubner, P. Lampe, J. Höchst, L. Baumgärtner, M. Mezini, and B. Freisleben, "Opportunistic Named Functions in Disruption-tolerant Emergency Networks," in *ACM International Conference on Computing Frontiers 2018 (ACM CF'18)*, Ischia, Italy: ACM, 2018, pp. 129–137 (cit. on pp. 6, 95).
- [20] C. Meurisch, J. Gedeon, A. Gogel, T. A. B. Nguyen, F. Kaup, F. Kohnhäuser, L. Baumgärtner, M. Schmittner, and M. Mühlhäuser, "Temporal Coverage Analysis of Router-based Cloudlets Using Human Mobility Patterns," in *2017 IEEE Global Communications Conference: Selected Areas in Communications: Internet of Things (GlobeCom 2017 SAC IoT)*, Singapore, Singapore: IEEE, 2017, pp. 1–6 (cit. on p. 6).
- [21] J. Höchst, L. Baumgärtner, M. Hollick, and B. Freisleben, "Unsupervised Traffic Flow Classification Using a Neural Autoencoder," in *42nd Annual IEEE Conference on Local Computer Networks (LCN 2017)*, Singapore, Singapore: IEEE, 2017, pp. 523–526 (cit. on p. 6).
- [22] M. Leinweber, T. Fober, M. Strickert, L. Baumgärtner, G. Klebe, B. Freisleben, and E. Hüllermeier, "CavSimBase: A Database for Large Scale Comparison of Protein Binding Sites," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 6, pp. 1423–1434, 2016 (cit. on p. 7).
- [23] M. Leinweber, L. Baumgärtner, M. Mernberger, T. Fober, E. Hüllermeier, G. Klebe, and B. Freisleben, "GPU-based Cloud Computing for Comparing the Structure of Protein Binding Sites," in *6th IEEE International Conference on Digital Ecosystems Technologies (DEST 2012)*, Campione d'Italia, Italy: IEEE, 2012, pp. 1–6 (cit. on p. 7).
- [24] A. Vahdat and D. Becker, "Epidemic Routing for Partially Connected Ad Hoc Networks," Duke University, Tech. Rep., Jul. 2000 (cit. on p. 11).
- [25] A. Lindgren, A. Doria, and O. Schelén, "Probabilistic Routing in Intermittently Connected Networks," *ACM SIGMOBILE mobile computing and communications review*, vol. 7, no. 3, pp. 19–20, 2003 (cit. on p. 11).
- [26] P. Gardner-Stephen, "The Serval Project: Practical Wireless Ad-Hoc Mobile Telecommunications," Flinders University, Adelaide, South Australia, Tech. Rep., Aug. 2011, pp. 1–29 (cit. on pp. 11, 48, 49).
- [27] P. Gardner-Stephen, R. Challans, J. Lakeman, A. Bettison, D. Gardner-Stephen, and M. Lloyd, "The Serval Mesh: A Platform for Resilient Communications in Disaster & Crisis," in *IEEE Global Humanitarian Technology Conference (GHTC 2013)*, IEEE, 2013, pp. 162–166 (cit. on pp. 11, 48, 76, 81).
- [28] P. Gardner-Stephen, A. Bettison, R. Challans, and J. Lakeman, "The Rational Behind The Serval Network Layer For Resilient Communications," *Journal of Computer Science*, vol. 9, no. 12, p. 1680, 2013 (cit. on pp. 11, 48, 76, 81, 104).
- [29] P. Gardner-Stephen, J. Lakeman, R. Challans, C. Wallis, A. Stulman, and Y. Haddad, "MeshMS: Ad Hoc Data Transfer within a Mesh Network," *International Journal of Communications, Network and System Sciences*, vol. 8, no. 5, pp. 496–504, 2012 (cit. on pp. 12, 48, 76, 81).

Bibliography

- [30] D. Johnson, N. Ntlatlapa, and C. Aichele, "A Simple Pragmatic Approach to Mesh Routing Using B.A.T.M.A.N.," in *2nd IFIP International Symposium on Wireless Communications and Information Technology in Developing Countries*, Pretoria, South Africa, 2008 (cit. on pp. 12, 81).
- [31] P. Jacquet, P. Muhlethaler, T. Clausen, A. Laouiti, A. Qayyum, and L. Viennot, "Optimized Link State Routing Protocol for Ad Hoc Networks," in *IEEE International Conference on Technology for the 21st Century*, IEEE, 2001, pp. 62–68 (cit. on p. 12).
- [32] C. Jackson and A. Barth, "Forcehttps: Protecting High-security Web Sites From Network Attacks," in *Proceeding of the 17th International Conference on World Wide Web*, ACM, 2008, pp. 525–534 (cit. on pp. 20, 141, 142).
- [33] Y. Song, C. Yang, and G. Gu, "Who is Peeping at Your Passwords at Starbucks? – To Catch An Evil Twin Access Point," in *IEEE/IFIP International Conference on Dependable Systems and Networks*, IEEE, 2010, pp. 323–332 (cit. on pp. 20, 141, 142).
- [34] E. Weingärtner, H. vom Lehn, and K. Wehrle, "A Performance Comparison of Recent Network Simulators," in *IEEE International Conference on Communications (ICC 2009)*, IEEE, 2009, pp. 1–5 (cit. on p. 33).
- [35] L. Hogue, P. Bouvry, and F. Guinand, "An Overview of MANET Simulation," *Electronic notes in Theoretical Computer Science*, vol. 150, no. 1, pp. 81–101, 2006 (cit. on p. 33).
- [36] M. Kropff, T. Krop, M. Hollick, P. S. Mogre, and R. Steinmetz, "A Survey on Real World and Emulation Testbeds for Mobile Ad Hoc Networks," in *2nd International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities*, IEEE, 2006, pp. 448–453 (cit. on pp. 33, 34).
- [37] K. N. Patel and R. h. Jhaveri, "A Survey on Emulation Testbeds for Mobile Ad-hoc Networks," *Procedia Computer Science*, vol. 45, pp. 581–591, 2015 (cit. on pp. 33, 34).
- [38] J. Ahrenholz, C. Danilov, T. R. Henderson, and J. H. Kim, "CORE: A Real-time Network Emulator," in *IEEE Military Communications Conference (MILCOM 2008)*, IEEE, 2008, pp. 1–7 (cit. on pp. 33, 34).
- [39] M. To, M. Cano, and P. Biba, "DOCKEMU—A Network Emulation Tool," in *IEEE 29th International Conference on Advanced Information Networking and Applications (WAINA 2015)*, IEEE, 2015, pp. 593–598 (cit. on pp. 33, 34).
- [40] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown, "Reproducible Network Experiments Using Container-based Emulation," in *8th International Conference on Emerging Networking Experiments and Technologies (CoNEXT 2012)*, Nice, France: ACM, 2012, pp. 253–264 (cit. on pp. 33, 35).
- [41] J. D. Britos, S. Arias, N. Echániz, G. Iribarren, L. Aimaretto, and G. Hirschfeld, "BATMAN Advanced Mesh Network Emulator," in *XXI Congreso Argentino de Ciencias de la Computación*, 2015, pp. 1–8 (cit. on p. 34).
- [42] R. Davoli, "VDE: Virtual Distributed Ethernet," in *International Conference on Testbeds and Research Infrastructures for the DEvelopment of NeTworks and COMmunities*, IEEE, 2005, pp. 213–220 (cit. on pp. 34, 37).

- [43] M. Pizzonia and M. Rimondini, "Netkit: Easy Emulation of Complex Networks on Inexpensive Hardware," in *4th International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities, ICST*, 2008, pp. 1–7 (cit. on p. 34).
- [44] T. Li, W. E. Thain Jr, and T. Fallon, "On the Use of Virtualization for Router Network Simulation," in *American Society for Engineering Education, ASEE*, 2010, pp. 9–16 (cit. on p. 34).
- [45] M. Pužar and T. Plagemann, "NEMAN: A Network Emulator for Mobile Ad-hoc Networks," in *8th International Conference on Telecommunications*, 2005, pp. 155–161 (cit. on p. 34).
- [46] S. Hemminger, "Network Emulation with NetEm," in *Australia's 6th National Linux Conference*, 2005, pp. 1–7 (cit. on pp. 34, 40).
- [47] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, *et al.*, "The Design and Implementation of Open vswitch," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, 2015, pp. 117–130 (cit. on p. 35).
- [48] S.-Y. Wang, "Comparison of SDN OpenFlow Network Simulator and Emulators: EstiNet vs. Mininet," in *IEEE Symposium on Computers and Communications (ISCC 2014)*, IEEE, 2014, pp. 1–6 (cit. on p. 35).
- [49] P. D. Pradeep and B. A. Kumar, "A Survey of Emergency Communication Network Architectures," *International Journal of u-and e-Service, Science and Technology*, vol. 8, no. 4, pp. 61–68, 2015 (cit. on pp. 49, 103, 115).
- [50] M. Berioli, N. Courville, and M. Werner, "Emergency Communications over Satellite: the WISECOM Approach," in *16Th IST Mobile and Wireless Communications Summit*, IEEE, 2007, pp. 1–5 (cit. on pp. 49, 115).
- [51] A. S. Cacciapuoti, F. Calabrese, M. Caleffi, G. Di Lorenzo, and L. Paura, "Human-mobility Enabled Wireless Networks for Emergency Communications during Special Events," *Pervasive and Mobile Computing*, vol. 9, no. 4, pp. 472–483, 2013 (cit. on pp. 49, 115).
- [52] W. Wang, W. Gao, X. Bai, T. Peng, G. Chuai, and W. Wang, "A Framework of Wireless Emergency Communications Based on Relaying and Cognitive Radio," in *IEEE 18th International Symposium on Personal, Indoor and Mobile Radio Communications*, IEEE, 2007, pp. 1–5 (cit. on pp. 49, 115).
- [53] M. Manic, D. Wijayasekara, K. Amarasinghe, J. Hewlett, K. Handy, C. Becker, B. Patterson, and R. Peterson, "Next Generation Emergency Communication Systems via Software Defined Networks," in *Third GENI Research and Educational Experiment Workshop*, IEEE, 2014, pp. 1–8 (cit. on pp. 49, 115).
- [54] V. Mayer-Schönberger, "Emergency Communications: The Quest for Interoperability in the United States and Europe," *John F. Kennedy School of Government, Harvard University*, 2002 (cit. on p. 49).
- [55] T. Pecorella, L. S. Ronga, F. Chiti, S. Jayousi, and L. Franck, "Emergency Satellite Communications: Research and Standardization Activities," *IEEE Communications Magazine*, vol. 53, no. 5, pp. 170–177, 2015 (cit. on pp. 49, 115).

Bibliography

- [56] H. Chenji and R. Stoleru, "Delay-tolerant Networks (DTNs) for Emergency Communications," *Advances in Delay-tolerant Networks (DTNs): Architecture and Enhanced Performance*, p. 105, 2014 (cit. on pp. 49, 103, 104, 115).
- [57] J. Thomas, J. Robble, and N. Modly, "Off-grid Communications with Android Meshing the Mobile World," in *IEEE Conference on Technologies for Homeland Security (HST 2012)*, IEEE, 2012, pp. 401–405 (cit. on pp. 49, 104, 115).
- [58] Y. Liu, D. R. Bild, D. Adrian, G. Singh, R. P. Dick, D. S. Wallach, and Z. M. Mao, "Performance and Energy Consumption Analysis of a Delay-tolerant Network for Censorship-resistant Communication," in *Proceedings of the 16th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, ACM, 2015, pp. 257–266 (cit. on pp. 49, 62).
- [59] H. Ntareme, M. Zennaro, and B. Pehrson, "Delay Tolerant Network on Smartphones: Applications for Communication Challenged Areas," in *Proceedings of the 3rd Extreme Conference on Communication*, ACM, 2011, pp. 14–21 (cit. on pp. 49, 115).
- [60] K. Heimerl, K. Ali, J. Blumenstock, B. Gawalt, and E. Brewer, "Expanding Rural Cellular Networks with Virtual Coverage," in *10th USENIX Symposium on Network Systems Design & Implementation*, 2013, pp. 283–296 (cit. on pp. 50, 115).
- [61] A. Battestini, V. Setlur, and T. Sohn, "A Large Scale Study of Text-messaging Use," in *12th International Conference on Human Computer Interaction with Mobile Devices and Services*, ACM, 2010, pp. 229–238 (cit. on p. 52).
- [62] G. Aloï, M. Di Felice, V. Loscri, P. Pace, and G. Ruggeri, "Spontaneous Smartphone Networks as a User-centric Solution for the Future Internet," *IEEE Communications Magazine*, vol. 52, no. 12, pp. 26–33, 2014 (cit. on p. 60).
- [63] E. Natsheh, A. B. Jantan, S. Khatun, and S. Shamala, "Adaptive Optimizing of Hello Messages in Wireless Ad-Hoc Networks," *Int. Arab J. Inf. Technol.*, vol. 4, no. 3, pp. 191–200, 2007 (cit. on p. 61).
- [64] M. B. Khalaf, A. Y. Al-Dubai, and W. Buchanan, "A New Adaptive Broadcasting Approach for Mobile Ad Hoc Networks," in *6th Conference on Wireless Advanced (WiAD 2010)*, IEEE, 2010, pp. 1–6 (cit. on p. 61).
- [65] S. H. Ahmed, S. H. Bouk, and D. Kim, "Adaptive Beaconing Schemes in VANETs: Hybrid Approach," in *International Conference on Information Networking (ICOIN 2015)*, IEEE, 2015, pp. 340–345 (cit. on p. 61).
- [66] R. Tahar, A. Dhraief, A. Belghith, H. Mathkour, and R. Braham, "Autonomous and Adaptive Beaconing Strategy for Multi-interfaced Wireless Mobile Nodes," *Wireless Communications and Mobile Computing*, vol. 16, no. 12, pp. 1625–1641, 2016 (cit. on p. 61).
- [67] A. Hess, E. Hyttiä, and J. Ott, "Efficient Neighbor Discovery in Mobile Opportunistic Networking Using Mobility Awareness," in *Sixth International Conference on Communication Systems and Networks (COMSNETS 2014)*, IEEE, 2014, pp. 1–8 (cit. on p. 61).
- [68] F. Peng, "A Novel Adaptive Mobility-aware MAC Protocol in Wireless Sensor Networks," *Wireless Personal Communications*, vol. 81, no. 2, pp. 489–501, 2015 (cit. on p. 62).

- [69] S. Lim, C. Yu, and C. R. Das, "RandomCast: An Energy-efficient Communication Scheme for Mobile Ad Hoc Networks," *IEEE Transactions on Mobile Computing*, vol. 8, no. 8, pp. 1039–1051, 2009 (cit. on p. 62).
- [70] J. A. B. Link, C. Wollgarten, S. Schupp, and K. Wehrle, "Perfect Difference Sets for Neighbor Discovery: Energy Efficient and Fair," in *Proceedings of the 3rd Extreme Conference on Communication: The Amazon Expedition*, ACM, 2011, p. 5 (cit. on p. 62).
- [71] G. Dán, N. Carlsson, and I. Chatzidrossos, "Efficient and Highly Available Peer Discovery: A Case for Independent Trackers and Gossiping," in *IEEE International Conference on Peer-to-Peer Computing (P2P 2011)*, IEEE, 2011, pp. 290–299 (cit. on p. 62).
- [72] R. Zheng, J. C. Hou, and L. Sha, "Asynchronous Wakeup for Ad Hoc Networks," in *Proceedings of the 4th ACM International Symposium on Mobile Ad Hoc Networking & Computing*, ACM, 2003, pp. 35–45 (cit. on p. 62).
- [73] P. S. Paul, B. C. Ghosh, K. De, S. Saha, S. Nandi, S. Saha, I. Bhattacharya, and S. Chakraborty, "On Design and Implementation of a Scalable and Reliable Sync System for Delay Tolerant Challenged Networks," in *8th International Conference on Communication Systems and Networks (COMSNETS 2016)*, IEEE, 2016, pp. 1–8 (cit. on p. 62).
- [74] B. Zhang, Y. Li, D. Jin, P. Hui, and Z. Han, "Social-Aware Peer Discovery for D2D Communications Underlying Cellular Networks," *IEEE Transactions on Wireless Communications*, vol. 14, no. 5, pp. 2426–2439, 2015 (cit. on p. 62).
- [75] W. Wang, V. Srinivasan, and M. Motani, "Adaptive Contact Probing Mechanisms for Delay Tolerant Applications," in *13th Annual ACM International Conference on Mobile Computing and Networking*, ACM, 2007, pp. 230–241 (cit. on p. 62).
- [76] S. Trifunovic, B. Distl, D. Schatzmann, and F. Legendre, "WiFi-Opp: Ad-hoc-less Opportunistic Networking," in *6th ACM Workshop on Challenged Networks*, ACM, 2011, pp. 37–42 (cit. on p. 62).
- [77] T. Clausen, C. Dearlove, and J. Dean, *Rfc 6130: Mobile ad hoc network (manet) neighborhood discovery protocol (nhdp)*, ietf, 2011 (cit. on p. 63).
- [78] A. D. Birrell and B. J. Nelson, "Implementing Remote Procedure Calls," *ACM Transactions on Computer Systems (TOCS 1084)*, vol. 2, no. 1, pp. 39–59, Feb. 1984 (cit. on p. 75).
- [79] K. Fall, "A Delay-tolerant Network Architecture for Challenged Internets," in *2003 ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Karlsruhe, Germany: ACM, 2003, pp. 27–34 (cit. on p. 75).
- [80] A. McMahon and S. Farrell, "Delay- and Disruption-Tolerant Networking," *IEEE Internet Computing*, vol. 13, pp. 82–87, 2009 (cit. on p. 75).
- [81] M. Kaweckı and R. O. Schoeneich, "Mobility-based Routing Algorithm in Delay Tolerant Networks," *EURASIP Journal on Wireless Communications and Networking*, vol. 2016, no. 1, pp. 1–9, 2016 (cit. on p. 75).

Bibliography

- [82] P. Gardner-Stephen and S. Palaniswamy, "Serval Mesh Software - WiFi Multi Model Management," in *Proceedings of the 1st International Conference on Wireless Technologies for Humanitarian Relief*, Amritapuri, Kollam, Kerala, India: ACM, 2011, pp. 71–77 (cit. on p. 75).
- [83] J. Tu and C. Stewart, "Replication for Predictability in a Java RPC Framework," in *IEEE International Conference on Autonomic Computing (ICAC 2015)*, IEEE, 2015, pp. 163–164 (cit. on p. 76).
- [84] P. Stuedi, A. Trivedi, B. Metzler, and J. Pfefferle, "DaRPC: Data Center RPC," in *ACM Symposium on Cloud Computing (SOCC 2014)*, Seattle, WA, USA: ACM, 2014, 15:1–15:13 (cit. on p. 76).
- [85] H. Chen, L. Shi, J. Sun, K. Li, and L. He, "A Fast RPC System for Virtual Machines," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 7, pp. 1267–1276, 2013 (cit. on p. 76).
- [86] N. Shyam, C. Harmer, and K. Beck, *Managing Remote Procedure Calls When a Server is Unavailable*, US Patent App. 12/610,049, May 2011 (cit. on p. 76).
- [87] A. Reinhardt, P. S. Mogre, and R. Steinmetz, "Lightweight Remote Procedure Calls for Wireless Sensor and Actuator Networks," in *IEEE International Conference on Pervasive Computing and Communications Workshops*, IEEE, 2011, pp. 172–177 (cit. on p. 76).
- [88] C. Shi, V. Lakafosis, M. H. Ammar, and E. W. Zegura, "Serendipity: Enabling Remote Computing Among Intermittently Connected Mobile Devices," in *13th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2012)*, Hilton Head, USA: ACM, 2012, pp. 145–154 (cit. on p. 77).
- [89] M. Chen, Y. Hao, Y. Li, C.-F. Lai, and D. Wu, "On the Computation Offloading at Ad Hoc Cloudlet: Architecture and Service Modes," *IEEE Communications Magazine*, vol. 53, no. 6, pp. 18–24, 2015 (cit. on p. 77).
- [90] Y. Zhang, D. Niyato, and P. Wang, "Offloading in Mobile Cloudlet Systems with Intermittent Connectivity," *IEEE Transactions on Mobile Computing*, vol. 14, no. 12, pp. 2516–2529, Dec. 2015, ISSN: 1536-1233 (cit. on p. 77).
- [91] Y. Lai, X. Gao, M. Liao, J. Xie, Z. Lin, and H. Zhang, "Data Gathering and Offloading in Delay Tolerant Mobile Networks," *Wireless Networks*, vol. 22, no. 3, pp. 959–973, 2016 (cit. on p. 77).
- [92] J. Schöning and G. Heidemann, "Image Based Spare Parts Reconstruction for Repairing Vital Infrastructure after Disasters," in *IEEE Global Humanitarian Technology Conference (GHTC '16)*, Seattle, USA: IEEE, 2016, pp. 225–232 (cit. on p. 88).
- [93] K. Aberer, S. Sathe, D. Chakraborty, A. Martinoli, G. Barrenetxea, B. Faltings, and L. Thiele, "OpenSense: Open Community Driven Sensing of the Environment," in *ACM SIGSPATIAL Int. Workshop on GeoStreaming*, ACM, 2010, pp. 39–42 (cit. on p. 90).
- [94] B. Maag, Z. Zhou, and L. Thiele, "W-Air: Enabling Personal Air Pollution Monitoring on Wearables," *Proceedings of ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 2, no. 1, p. 24, 2018 (cit. on p. 90).

- [95] N. Castell, M. Kobernus, H.-Y. Liu, P. Schneider, W. Lahoz, A. J. Berre, and J. Noll, "Mobile Technologies and Services for Environmental Monitoring: The Citi-Sense-MOB Approach," *Urban Climate*, vol. 14, pp. 370–382, 2015 (cit. on p. 90).
- [96] M. Gerboles, L. Spinelle, A. Kotsev, M. Signorini, and L. Srl, "AirSenseEUR: An Open-Designed Multi-Sensor Platform for Air Quality Monitoring," in *4th Scientific Meeting EuNetAir*, 2015, pp. 3–5 (cit. on p. 90).
- [97] P. Sikka, P. Corke, L. Overs, P. Valencia, and T. Wark, "Fleck-a Platform for Real-world Outdoor Sensor Networks," in *3rd International Conference on Intelligent Sensors, Sensor Networks and Information*, IEEE, 2007, pp. 709–714 (cit. on p. 90).
- [98] M. T. Lazarescu, "Design and Field Test of a WSN Platform Prototype for Long-term Environmental Monitoring," *Sensors*, vol. 15, no. 4, pp. 9481–9518, 2015 (cit. on p. 90).
- [99] C. Llamas, M. A. González, C. Hernández, and J. Vegas, "Open Source Hardware Based Sensor Platform Suitable for Human Gait Identification," *Pervasive and Mobile Computing*, vol. 38, pp. 154–165, 2017 (cit. on p. 90).
- [100] J. A. Wirwahn and T. Bartoschek, "Usability Engineering For Successful Open Citizen Science," in *Free and Open Source Software for Geospatial (FOSS4G) Conference Proceedings*, vol. 15, 2015, p. 54 (cit. on p. 90).
- [101] N. Fröschle, "Engineering of New Participation Instruments Exemplified by Electromobility, Particulate Matter and Clean Air Policy-Making," *HMD Praxis der Wirtschaftsinformatik*, vol. 54, no. 4, pp. 502–517, 2017 (cit. on p. 90).
- [102] C. Pham, A. Rahim, and P. Cousin, "WAZIUP: A Low-Cost Infrastructure for Deploying IoT in Developing Countries," in *International Conference on e-Infrastructure and e-Services for Developing Countries*, Springer, 2016, pp. 135–144 (cit. on p. 90).
- [103] Y. Mao, J. Wang, B. Sheng, and F. Wu, "Building Smartphone Ad-hoc Networks with Long-range Radios," in *34th International Conference on Computing and Communications*, IEEE, 2015, pp. 1–8 (cit. on p. 90).
- [104] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, IEEE, 2016, pp. 2818–2826 (cit. on p. 95).
- [105] P. Viola and M. J. Jones, "Robust Real-time Face Detection," *International Journal of Computer Vision*, vol. 57, no. 2, pp. 137–154, 2004 (cit. on pp. 104, 106).
- [106] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object Detection with Discriminatively Trained Part-based Models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 9, pp. 1627–1645, 2010 (cit. on p. 104).
- [107] J. Cheney, B. Klein, A. K. Jain, and B. F. Klare, "Unconstrained Face Detection: State of the Art Baseline and Challenges," in *International Conference on Biometrics (ICB 2015)*, IEEE, 2015, pp. 229–236 (cit. on pp. 104, 106, 108).

Bibliography

- [108] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller, "Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments," University of Massachusetts, Amherst, Tech. Rep., 2007 (cit. on p. 104).
- [109] L. Wolf, T. Hassner, and I. Maoz, "Face Recognition in Unconstrained Videos with Matched Background Similarity," in *IEEE Conference on Computer Vision and Pattern Recognition*, IEEE, 2011, pp. 529–534 (cit. on p. 104).
- [110] B. F. Klare, B. Klein, E. Taborsky, A. Blanton, J. Cheney, K. Allen, P. Grother, A. Mah, and A. K. Jain, "Pushing the Frontiers of Unconstrained Face Detection and Recognition: ARPA Janus Benchmark A," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, IEEE, 2015, pp. 1931–1939 (cit. on p. 104).
- [111] K. Imaizumi and V. G. Moshnyaga, "Network-based Face Recognition on Mobile Devices," in *IEEE 3rd International Conference on Consumer Electronics*, IEEE, 2013, pp. 406–409 (cit. on p. 104).
- [112] T. Soyata, R. Muraleedharan, C. Funai, M. Kwon, and W. Heinzelman, "Cloud-vision: Real-time Face Recognition Using a Mobile-cloudlet-cloud Acceleration Architecture," in *IEEE Symposium on Computers and Communications (ISCC 2012)*, IEEE, 2012, pp. 59–66 (cit. on p. 104).
- [113] H. Feng, B. Wang, C. Zhang, B. Yu, W. Hwang, J.-J. Han, C. Choi, and H. Wang, "A Fast Multi-view Face Detector for Mobile Phone," in *IEEE International Conference on Image Processing (ICIP 2016)*, IEEE, 2016, pp. 3219–3223 (cit. on p. 104).
- [114] H. L. Akin, N. Ito, A. Jacoff, A. Kleiner, J. Pellenz, and A. Visser, "Robocup Rescue Robot and Simulation Leagues," *AI magazine*, vol. 34, no. 1, p. 78, 2012 (cit. on p. 114).
- [115] D. Fan, B. Li, and M. Chen, "Design of Global Emergency Mobile Communication System Based on TDRSS," in *6th International Conference on Electronics Information and Emergency Communication (ICEIEC 2016)*, IEEE, 2016, pp. 322–325 (cit. on p. 115).
- [116] K. Igarashi, K. Umeno, M. Okada, and M. Kikuchi, "Study on Emergency Message Communication System for Ensuring Safety in Antarctica under Extremely Severe Environments," in *International Conference on Smart Green Technology in Electrical and Information Systems (ICSGTEIS 2016)*, IEEE, 2016, pp. 116–119 (cit. on p. 115).
- [117] K. Heimerl and T. S. Parikh, "How Users Understand Cellular Infrastructure," University of California, Berkeley, Tech. Rep., Apr. 2012 (cit. on p. 115).
- [118] C. Köbel, W. B. Garcia, and J. Habermann, "A Survey on Wireless Mesh Network Applications in Rural Areas and Emerging Countries," in *IEEE Global Humanitarian Technology Conference (GHTC 2013)*, IEEE, 2013, pp. 389–394 (cit. on p. 115).

- [119] D. Reina, J. Coca, M. Askalani, S. Toral, F. Barrero, E. Asimakopoulou, S. Sotiriadis, and N. Bessis, "A Survey on Ad Hoc Networks for Disaster Scenarios," in *International Conference on Intelligent Networking and Collaborative Systems (INCoS 2014)*, IEEE, 2014, pp. 433–438 (cit. on p. 115).
- [120] Briar, <https://briarproject.org/>, (Accessed on 06/25/2016) (cit. on p. 115).
- [121] Firechat, <http://opengarden.com/about-firechat>, (Accessed on 06/25/2016) (cit. on p. 115).
- [122] Z. Lu, G. Cao, and T. La Porta, "Networking Smartphones for Disaster Recovery," in *IEEE International Conference on Pervasive Computing and Communications (PerCom 2016)*, IEEE, 2016, pp. 1–9 (cit. on p. 115).
- [123] H. Nishiyama, M. Ito, and N. Kato, "Relay-by-smartphone: Realizing Multihop Device-to-Device Communications," *IEEE Communication Magazine*, vol. 52, no. 4, pp. 56–65, Apr. 2014 (cit. on p. 115).
- [124] R. Murphy, S. Tadokoro, D. Nardi, A. Joacoff, P. Fiorini, H. Choset, and A. Erkmén, *Search and Rescue Robotics, Fundamental Problems and Open Issues in Handbook of Robotics*, eds. Siciliano, Bruno; Khatib, Oussama, 2008 (cit. on p. 115).
- [125] R. R. Murphy, "Trial by fire [rescue robots]," *IEEE Robotics & Automation Magazine*, vol. 11, no. 3, pp. 50–61, 2004 (cit. on p. 115).
- [126] T. Yoshida, K. Nagatani, S. Tadokoro, T. Nishimura, and E. Koyanagi, "Improvements to the Rescue Robot Quince Toward Future Indoor Surveillance Missions in the Fukushima Daiichi Nuclear Power Plant," in *Field and Service Robotics*, Springer, 2014, pp. 19–32 (cit. on p. 115).
- [127] A. Hart, N. Pezeshkian, and H. Nguyen, "Mesh Networking Optimized for Robotic Teleoperation," Space and Naval Warfare Systems Center San Diego, CA, Tech. Rep., 2012 (cit. on p. 115).
- [128] C. Luo, P. Ward, S. Cameron, G. Parr, and S. McClean, "Communication Provision for a Team of Remotely Searching UAVs: A Mobile Relay Approach," in *2012 IEEE Globecom Workshops*, IEEE, 2012, pp. 1544–1549 (cit. on p. 115).
- [129] E. F. Flushing, M. Kudelski, L. M. Gambardella, and G. A. Di Caro, "Connectivity-aware Planning of Search and Rescue Missions," in *IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR 2013)*, IEEE, 2013, pp. 1–8 (cit. on p. 115).
- [130] J. A. Dias, J. N. Isento, V. N. Soares, and J. J. Rodrigues, "Impact of Scheduling and Dropping Policies on the Performance of Vehicular Delay-tolerant Networks," in *IEEE International Conference on Communications (ICC 2011)*, IEEE, 2011, pp. 1–5 (cit. on p. 115).
- [131] M. Frassl, M. Lichtenstern, M. Khider, and M. Angermann, "Developing a System for Information Management in Disaster Relief-Methodology and Requirements," in *7th International Conference on Information Systems for Crisis Response And Management (ISCRAM'10)*, vol. 1, 2010 (cit. on p. 115).
- [132] A. M. Khaleghi, D. Xu, S. Minaeian, M. Li, Y. Yuan, J. Liu, Y.-J. Son, C. Vo, A. Mousavian, and J.-M. Lien, "A Comparative Study of Control Architectures in UAV/UGV-based Surveillance System," in *IIE Annual Conference. Proceedings*, Institute of Industrial Engineers-Publisher, 2014, p. 3455 (cit. on p. 115).

Bibliography

- [133] H.-B. Kuntze, C. W. Frey, I. Tchouchenkov, B. Staehle, E. Rome, K. Pfeiffer, A. Wenzel, and J. Wöllenstein, "SENEKA-Sensor Network with Mobile Robots for Disaster Management," in *IEEE Conference on Technologies for Homeland Security (HST'12)*, IEEE, 2012, pp. 406–410 (cit. on p. 115).
- [134] H.-B. Kuntze, C. Frey, T. Emter, J. Petereit, I. Tchouchenkov, T. Mueller, M. Tittel, R. Worst, K. Pfeiffer, M. Walter, *et al.*, "Situation Responsive Networking of Mobile Robots for Disaster Management," in *Proceedings of ISR/Robotik 2014; 41st International Symposium on Robotics*, VDE, 2014, pp. 1–8 (cit. on p. 115).
- [135] S. Kohlbrecher, J. Meyer, T. Graber, K. Petersen, O. von Stryk, and U. Klingauf, "Hector Open Source Modules for Autonomous Mapping and Navigation with Rescue Robots," in *Proceedings of 17th RoboCup International Symposium*, 2013 (cit. on p. 116).
- [136] J. Kuhn, C. Reinl, and O. von Stryk, "Predictive Control for Multi-Robot Observation of Multiple Moving Targets Based on Discrete-Continuous Linear Models," in *Proceedings of the 18th IFAC World Congress*, 2011, pp. 257–262 (cit. on p. 116).
- [137] T. Ritter, J. Euler, S. Ulbrich, and O. von Stryk, "Decentralized Dynamic Data-driven Monitoring of Atmospheric Dispersion Processes," *Procedia Computer Science*, vol. 80, pp. 919–930, 2016 (cit. on p. 116).
- [138] A. Bemporad and M. Morari, "Control of Systems Integrating Logic, Dynamics, and Constraints," *Automatica*, vol. 35, pp. 407–427, 1999 (cit. on p. 116).
- [139] M. Schmittner, A. Asadi, and M. Hollick, "SEMUD: Secure Multi-hop Device-to-Device Communication for 5G Public Safety Networks," in *IFIP Networking Conference (Networking'17)*, Stockholm, Sweden, 2017 (cit. on p. 118).
- [140] T. A. B. Nguyen, C. Meurisch, S. Niemczyk, D. Böhnstedt, K. Geihs, M. Mühlhäuser, and R. Steinmetz, "Adaptive Task-Oriented Message Template for In-Network Processing," in *International Conference on Networked Systems (NetSys'17)*, IEEE, 2017, pp. 1–8 (cit. on p. 120).
- [141] C. Meurisch, T. A. B. Nguyen, J. Gedeon, F. Kohnhäuser, M. Schmittner, S. Niemczyk, S. Wullkotte, and M. Mühlhäuser, "Upgrading Wireless Home Routers as Emergency Cloudlet and Secure DTN Communication Bridge," in *26th International Conference on Computer Communications and Networks (ICCCN'17): Posters*, IEEE, 2017 (cit. on p. 120).
- [142] C. Meurisch, T. A. B. Nguyen, S. Wullkotte, S. Niemczyk, Kohnhäuser, and M. Mühlhäuser, "NICER911: Ad-hoc Communication and Emergency Services Using Networking Smartphones and Wireless Home Routers," in *18th International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'17): Poster*, ACM, 2017 (cit. on p. 121).
- [143] A. Jacoff, R. Sheh, A.-M. Virts, T. Kimura, J. Pellenz, S. Schwertfeger, and J. Suthakorn, "Using Competitions to Advance the Development of Standard Test Methods for Response Robots," in *Proceedings of the Workshop on Performance Metrics for Intelligent Systems*, ACM, 2012, pp. 182–189 (cit. on p. 121).

- [144] T. Hossmann, P. Carta, D. Schatzmann, F. Legendre, P. Gunningberg, and C. Rohner, "Twitter in Disaster Mode: Security Architecture," in *ACM Special Workshop on Internet and Disasters*, ACM, 2011, p. 7 (cit. on p. 128).
- [145] E. A. Panaousis, T. A. Ramrekha, C. Politis, and G. P. Millar, "Secure Decentralised Ubiquitous Networking for Emergency Communications," in *International Conference on Telecommunications and Multimedia (TEMU 2012)*, IEEE, 2012, pp. 233–238 (cit. on p. 128).
- [146] M. Puzar, T. Plagemann, and Y. Roudier, "Security and Privacy Issues in Middleware for Emergency and Rescue Applications," in *Second International Conference on Pervasive Computing Technologies for Healthcare (PervasiveHealth 2008)*, IEEE, 2008, pp. 89–92 (cit. on p. 128).
- [147] S. G. Weber, Y. Kalev, S. Ries, and M. Mühlhäuser, "MundoMessage: Enabling Trustworthy Ubiquitous Emergency Communication," in *Proceedings of the 5th International Conference on Ubiquitous Information Management and Communication*, ACM, 2011, p. 29 (cit. on p. 128).
- [148] H. Zhu, X. Lin, R. Lu, X. Shen, D. Xing, and Z. Cao, "An Opportunistic Batch Bundle Authentication Scheme for Energy Constrained DTNs," in *IEEE INFOCOM*, 2010 (cit. on p. 128).
- [149] J. Burgess, G. D. Bissias, M. D. Corner, and B. N. Levine, "Surviving Attacks on Disruption-tolerant Networks Without Authentication," in *Proceedings of the 8th ACM international symposium on Mobile ad hoc networking and computing (MobiHoc 2007)*, ACM, 2007, pp. 61–70 (cit. on pp. 128, 129).
- [150] A. Kate, G. M. Zaverucha, and U. Hengartner, "Anonymity and Security in Delay Tolerant Networks," in *IEEE SecureComm*, IEEE, 2007, pp. 504–513 (cit. on p. 128).
- [151] D. Ma and G. Tsudik, "Security and Privacy in Emerging Wireless Networks [Invited Paper]," *IEEE Wireless Communications*, 2010 (cit. on p. 128).
- [152] R. Chen, F. Bao, M. Chang, and J.-H. Cho, "Dynamic Trust Management for Delay Tolerant Networks and Its Application to Secure Routing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 5, pp. 1200–1210, 2014 (cit. on p. 128).
- [153] H. Zhu, S. Du, Z. Gao, M. Dong, and Z. Cao, "A probabilistic Misbehavior Detection Scheme Toward Efficient Trust Establishment in Delay-tolerant Networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 1, pp. 22–32, 2014 (cit. on p. 128).
- [154] J. Luo, J.-P. Hubaux, and P. T. Eugster, "Dictate: Distributed Certification Authority with Probabilistic Freshness for Ad Hoc Networks," *IEEE Transactions on Dependable and Secure Computing*, vol. 2, no. 4, pp. 311–323, 2005 (cit. on p. 128).
- [155] Q. Li, W. Gao, S. Zhu, and G. Cao, "To Lie or to Comply: Defending Against Flood Attacks in Disruption Tolerant Networks," *IEEE Transactions on Dependable and Secure Computing*, vol. 10, no. 3, pp. 168–182, 2013 (cit. on p. 129).
- [156] F. C. Lee, W. Goh, and C. K. Yeo, "A Queuing Mechanism to Alleviate Flooding Attacks in Probabilistic Delay Tolerant Networks," in *Sixth Advanced International Conference on Telecommunications (AICT 2010)*, IEEE, 2010, pp. 329–334 (cit. on p. 132).

Bibliography

- [157] A. Keränen, J. Ott, and T. Kärkkäinen, “The ONE Simulator for DTN Protocol Evaluation,” in *Proceedings of the 2nd international conference on simulation tools and techniques (ICST SIMUTools)*, 2009, p. 55 (cit. on p. 133).
- [158] W. Enck, M. Ongtang, and P. McDaniel, “On Lightweight Mobile Phone Application Certification,” in *Proceedings of the 16th ACM Conference on Computer and Communications Security*, ACM, 2009, pp. 235–245 (cit. on pp. 138, 141, 155).
- [159] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, “Android Permissions Demystified,” in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, ACM, 2011, pp. 627–638 (cit. on pp. 138, 141, 142, 155, 159, 160).
- [160] A. Porter Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, “Android Permissions: User Attention, Comprehension, and Behavior,” UC Berkeley, Tech. Rep., 2012 (cit. on pp. 138, 141, 153).
- [161] L. Davi, A. Dmitrienko, A. Sadeghi, and M. Winandy, “Privilege Escalation Attacks on Android,” in *Proceedings of the 13th International Conference on Information Security*, Springer, 2011, pp. 346–360 (cit. on pp. 138, 141).
- [162] S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, A. Sadeghi, and B. Shastri, “Towards Taming Privilege-Escalation Attacks on Android,” in *Proceedings of the 19th Network and Distributed System Security Symposium (NDSS)*, vol. 17, 2012, p. 19 (cit. on pp. 138, 141).
- [163] G. Portokalidis, P. Homburg, K. Anagnostakis, and H. Bos, “Paranoid Android: Versatile Protection for Smartphones,” in *Proceedings of the 26th Annual Computer Security Applications Conference*, ACM, Dec. 2010, pp. 347–356 (cit. on p. 138).
- [164] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, “TaintDroid: An Information-flow Tracking System For Realtime Privacy Monitoring on Smartphones,” in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, USENIX Association, 2010 (cit. on pp. 138, 141, 144).
- [165] W. Enck, D. Ocateau, P. McDaniel, and S. Chaudhuri, “A Study of Android Application Security,” in *Proceedings of the 20th USENIX Conference on Security*, 2011 (cit. on pp. 138, 142, 146).
- [166] M. Marlinspike, “More Tricks For Defeating SSL In Practice,” in *Black Hat USA*, 2009 (cit. on pp. 140, 142).
- [167] —, “New Tricks for Defeating SSL in Practice,” in *Black Hat Europe*, 2009 (cit. on pp. 140, 142).
- [168] M. Nauman, S. Khan, and X. Zhang, “Apex: Extending Android Permission Model And Enforcement With User-defined Runtime Constraints,” in *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, ACM, 2010 (cit. on p. 141).
- [169] A. Egners, B. Marschollek, and U. Meyer, “Messing with Android’s Permission Model,” in *Proceedings of the IEEE TrustCom*, IEEE, 2012, pp. 1–22 (cit. on p. 141).
- [170] T. Vidas, D. Votipka, and N. Christin, “All Your Droid Are Belong To Us: A Survey Of Current Android Attacks,” in *Proceedings of the 5th USENIX Workshop on Offensive Technologies*, 2011, pp. 10–10 (cit. on p. 142).

- [171] A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, S. Dolev, and C. Glezer, "Google Android: A Comprehensive Security Assessment," *Security & Privacy, IEEE*, vol. 8, no. 2, pp. 35–44, 2010 (cit. on p. 142).
- [172] W. Enck, M. Ongtang, and P. McDaniel, "Understanding Android Security," in *Proceedings of the IEEE International Conference on Security & Privacy*, IEEE, 2009, pp. 50–57 (cit. on p. 142).
- [173] P. McDaniel and W. Enck, "Not So Great Expectations: Why Application Markets Haven't Failed Security," *IEEE Security & Privacy*, vol. 8, no. 5, pp. 76–78, 2010 (cit. on p. 142).
- [174] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang, "Hey, You, Get Off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets," in *Proceedings of the 19th Annual Network and Distributed System Security Symposium (NDSS 2012)*, 2012 (cit. on p. 142).
- [175] D. Shin and R. Lopes, "An Empirical Study of Visual Security Cues to Prevent The SSLstripping Attack," in *Proceedings of the 27th Annual Computer Security Applications Conference*, ACM, 2011, pp. 287–296, ISBN: 1450306721 (cit. on p. 142).
- [176] S. Egelman, L. Cranor, and J. Hong, "You've Been Warned: An Empirical Study of the Effectiveness of Web Browser Phishing Warnings," in *Proceedings of the 26th Annual SIGCHI Conference on Human Factors in Computing Systems*, ACM, 2008, pp. 1065–1074 (cit. on pp. 142, 153).
- [177] J. Sunshine, S. Egelman, H. Almuhiemedi, N. Atri, and L. Cranor, "Crying Wolf: An Empirical Study of SSL Warning Effectiveness," in *Proceedings of the 18th USENIX Security Symposium*, 2009, pp. 399–416 (cit. on pp. 142, 153).
- [178] A. Sotirakopoulos and K. Hawkey, "'I Did it Because I Trusted You': Challenges With The Study Environment Biasing Participant Behaviours," in *Proceedings of the 6th Symposium on Usable Privacy and Security*, 2010 (cit. on p. 154).
- [179] A. Sotirakopoulos, K. Hawkey, and K. Beznosov, "On the Challenges in Usable Security Lab Studies: Lessons Learned From Replicating a Study on SSL Warnings," in *Proceedings of the 7th Symposium on Usable Privacy and Security*, Jul. 2011 (cit. on p. 154).
- [180] F. Maggi, A. Valdi, and S. Zanero, "Andrototal: A Flexible, Scalable Toolbox and Service for Testing Mobile Malware Detectors," in *Proceedings of the Third ACM Workshop on Security and Privacy in Smartphones & Mobile Devices*, ACM, 2013, pp. 49–54 (cit. on pp. 159, 161).
- [181] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, "Riskranker: Scalable and Accurate Zero-day Android Malware Detection," in *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, ACM, 2012, pp. 281–294 (cit. on pp. 159, 160).
- [182] N. Viennot, E. Garcia, and J. Nieh, "A Measurement Study of Google Play," in *The 2014 ACM International Conference on Measurement and Modeling of Computer Systems*, ACM, 2014, pp. 221–233 (cit. on pp. 159, 160, 164).
- [183] J. Kim, Y. Yoon, K. Yi, J. Shin, and S. Center, "ScanDal: Static Analyzer for Detecting Privacy Leaks in Android Applications," in *Proceedings of the 2012 Workshop on Security Technologies*, IEEE, 2012 (cit. on pp. 159, 160).

Bibliography

- [184] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "TaintDroid: An Information Flow Tracking System for Real-time Privacy Monitoring on Smartphones," *ACM Transactions on Computer Systems (TOCS 2014)*, vol. 57, no. 3, pp. 99–106, 2014 (cit. on p. 159).
- [185] A. Jindal, A. Pathak, Y. C. Hu, and S. Midkiff, "On Death, Taxes, and Sleep Disorder Bugs in Smartphones (HotPower 2013)," in *Proceedings of the Workshop on Power-Aware Computing and Systems*, Farmington, Pennsylvania: ACM, 2013, pp. 1–5 (cit. on p. 159).
- [186] A. Desnos and G. Gueguen, "Android: From Reversing to Decompilation," in *Proceedings of Black Hat Abu Dhabi*, 2011. [Online]. Available: <https://github.com/androguard/androguard> (cit. on pp. 160–162).
- [187] M. Egele, D. Brumley, Y. Fratantonio, and C. Kruegel, "An Empirical Study of Cryptographic Misuse in Android Applications," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, ACM, 2013, pp. 73–84 (cit. on pp. 160, 173).
- [188] A. Bartel, J. Klein, M. Monperrus, and Y. le Traon, "Static Analysis for Extracting Permission Checks of a Large Scale Framework: The Challenges and Solutions for Analyzing Android," *IEEE Transactions on Software Engineering*, vol. 40, no. 6, pp. 617–632, Jun. 2014 (cit. on p. 160).
- [189] Y. Feng, S. Anand, I. Dillig, and A. Aiken, "Apposcopy: Semantics-based Detection of Android Malware Through Static Analysis," in *2014 Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*, Hong Kong, China: ACM, 2014, pp. 576–587 (cit. on p. 160).
- [190] É. Payet and F. Spoto, "Static Analysis of Android Programs," *Information and Software Technology*, vol. 54, no. 11, pp. 1192–1201, 2012 (cit. on p. 160).
- [191] A.-D. Schmidt, R. Bye, H.-G. Schmidt, J. Clausen, O. Kiraz, K. A. Yuksel, S. A. Camtepe, and S. Albayrak, "Static Analysis of Executables for Collaborative Malware Detection on Android," in *IEEE International Conference on Communications*, IEEE, 2009, pp. 1–5 (cit. on p. 160).
- [192] J. Hoffmann, M. Ussath, T. Holz, and M. Spreitzenbarth, "Slicing Droids: Program Slicing for Smali Code," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, ACM, 2013, pp. 1844–1851 (cit. on p. 160).
- [193] L. Weichselbaum, M. Neugschwandtner, M. Lindorfer, Y. Fratantonio, V. van der Veen, and C. Platzer, "Andrubis: Android Malware Under The Magnifying Glass," *Vienna University of Technology, Tech. Rep. TRISECLAB-0414-001*, 2014 (cit. on p. 161).
- [194] L. Ravindranath, S. Nath, J. Padhye, and H. Balakrishnan, "Automatic and Scalable Fault Detection for Mobile Applications," in *2014 Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)*, Bretton Woods, New Hampshire, USA: ACM, 2014, pp. 190–203 (cit. on pp. 175, 176).

- [195] L. Ravindranath, J. Padhye, S. Agarwal, R. Mahajan, I. Obermiller, and S. Shayan-deh, "AppInsight: Mobile App Performance Monitoring in the Wild," in *2012 Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation (OSDI)*, Hollywood, CA, USA, 2012, pp. 107–120 (cit. on p. 175).
- [196] V. Rastogi, Y. Chen, and X. Jiang, "DroidChameleon: Evaluating Android Anti-malware Against Transformation Attacks," in *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, Hangzhou, China: ACM, 2013, pp. 329–334 (cit. on pp. 175, 181).
- [197] S. Hao, B. Liu, S. Nath, W. G. Halfond, and R. Govindan, "PUMA: Programmable UI-Automation for Large Scale Dynamic Analysis of Mobile Apps," in *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services*, ACM, 2014, pp. 204–217 (cit. on pp. 175, 176).
- [198] Genymobile. (2015). Genymotion, [Online]. Available: <http://www.genymotion.com> (visited on 02/19/2015) (cit. on pp. 175, 176).
- [199] Manymo LLC. (2015). Manymo, [Online]. Available: <https://www.manymo.com> (visited on 02/19/2015) (cit. on pp. 175, 176).
- [200] Bitbar. (2015). Testdroid, [Online]. Available: <http://testdroid.com> (visited on 02/19/2015) (cit. on pp. 175, 176).
- [201] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones," *ACM Transactions on Computer Systems (TOCS 2014)*, vol. 32, no. 2, 5:1–5:29, 2014 (cit. on p. 175).
- [202] D. Wu and R. Chang, "Analyzing Android Browser Apps for file:// Vulnerabilities," in *Information Security*, vol. 8783, Springer International Publishing, 2014, pp. 345–363 (cit. on p. 175).
- [203] A. Pathak, Y. C. Hu, and M. Zhang, "Where is the Energy Spent Inside my App? Fine Grained Energy Accounting on Smartphones with Eprof," in *Proceedings of the 7th ACM European Conference on Computer Systems*, ACM, 2012, pp. 29–42 (cit. on p. 175).
- [204] R. Mittal, A. Kansal, and R. Chandra, "Empowering Developers to Estimate App Energy Consumption," in *2012 Proceedings of the 18th Annual International Conference on Mobile Computing and Networking (Mobicom)*, Istanbul, Turkey: ACM, 2012, pp. 317–328 (cit. on p. 175).
- [205] S. Hao, D. Li, W. G. J. Halfond, and R. Govindan, "Estimating Mobile Application Energy Consumption Using Program Analysis," in *Proceedings of the 2013 International Conference on Software Engineering (ICSE)*, San Francisco, CA, USA: IEEE, 2013, pp. 92–101 (cit. on p. 175).
- [206] D. Amalfitano, A. R. Fasolino, P. Tramontana, S. De Carmine, and A. M. Memon, "Using GUI Ripping for Automated Testing of Android Applications," in *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, Essen, Germany: ACM, 2012, pp. 258–261 (cit. on pp. 175, 176).

Bibliography

- [207] G. Hu, X. Yuan, Y. Tang, and J. Yang, "Efficiently, Effectively Detecting Mobile App Bugs with AppDoctor," in *Proceedings of the Ninth European Conference on Computer Systems (EuroSys 2014)*, Amsterdam, The Netherlands: ACM, 2014, 18:1–18:15 (cit. on pp. 175, 176).
- [208] A. Machiry, R. Tahiliani, and M. Naik, "Dynodroid: An Input Generation System for Android Apps," in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, Saint Petersburg, Russia: ACM, 2013, pp. 224–234 (cit. on p. 175).
- [209] C.-J. M. Liang, N. Lane, N. Brouwers, L. Zhang, B. Karlsson, R. Chandra, and F. Zhao, "Contextual Fuzzing: Automated Mobile App Testing Under Dynamic Device and Environment Conditions," Microsoft Research, 2013 (cit. on p. 175).
- [210] M. Bierma, E. Gustafson, J. Erickson, D. Fritz, and Y. R. Choe, "Andlantis: Large-scale Android Dynamic Analysis," in *Proceedings of the 3rd IEEE Mobile Security Technologies Workshop (MoST 2014)*, IEEE, 2014 (cit. on pp. 175, 176, 179).
- [211] R. Mahmood, N. Esfahani, T. Kacem, N. Mirzaei, S. Malek, and A. Stavrou, "A Whitebox Approach for Automated Security Testing of Android Applications on the Cloud," in *7th International Workshop on Automation of Software Test (AST 2012)*, IEEE, 2012, pp. 22–28 (cit. on p. 176).
- [212] O. Starov and S. Vilkomir, "Integrated TaaS Platform for Mobile Development: Architecture Solutions," in *8th International Workshop on Automation of Software Test (AST 2013)*, IEEE, 2013, pp. 1–7 (cit. on p. 176).
- [213] Perfecto Mobile. (2015). Perfecto Mobile, [Online]. Available: <http://www.perfectomobile.com> (visited on 02/19/2015) (cit. on p. 176).
- [214] Keynote. (2015). Mobile Testing, [Online]. Available: <http://www.keynote.com/solutions/testing/%5Clinebreak%20mobile-testing> (visited on 02/19/2015) (cit. on p. 176).
- [215] Apkudo. (2015). Apkudo, [Online]. Available: <http://www.apkudo.com> (visited on 02/19/2015) (cit. on p. 176).
- [216] N. Viennot, E. Garcia, and J. Nieh, "A Measurement Study of Google Play," in *ACM International Conference on Measurement and Modeling of Computer Systems*, ACM, 2014, pp. 221–233 (cit. on pp. 176, 177).
- [217] B. F. Dolan-Gavitt, J. Hodosh, P. Hulin, T. Leek, and R. Whelan, "Repeatable Reverse Engineering for the Greater Good with PANDA," *Technical Report: CUCS-023-14*, 2014 (cit. on pp. 176, 179).
- [218] E. Zitzler, M. Laumanns, L. Thiele, E. Zitzler, E. Zitzler, L. Thiele, and L. Thiele, "SPEA2: Improving the Strength Pareto Evolutionary Algorithm," *Technical Report 103*, 2001 (cit. on p. 180).
- [219] S. Garfinkel, *PGP: Pretty Good Privacy*. O'Reilly Media, 1995 (cit. on p. 203).
- [220] N. J. Al Fardan and K. G. Paterson, "Lucky Thirteen: Breaking the TLS and DTLS Record Protocols," in *IEEE Symposium on Security and Privacy*, IEEE, 2013, pp. 526–540 (cit. on p. 203).

- [221] H. K. Lee, T. Malkin, and E. Nahum, "Cryptographic Strength of SSL/TLS Servers: Current and Recent Practices," in *7th ACM SIGCOMM Conference on Internet Measurement*, ACM, 2007, pp. 83–92 (cit. on p. 203).
- [222] A. Klein, "Attacks on the RC4 Stream Cipher," *Designs, Codes and Cryptography*, vol. 48, no. 3, pp. 269–286, 2008 (cit. on p. 203).
- [223] M. Stevens, A. Sotirov, J. Appelbaum, A. Lenstra, D. Molnar, D. A. Osvik, and B. De Weger, "Short Chosen-prefix Collisions for MD5 and the Creation of a Rogue CA Certificate," in *Advances in Cryptology-CRYPTO 2009*, Springer, 2009, pp. 55–69 (cit. on p. 203).
- [224] P. Eckersley and J. Burns, "Is the SSLiverse a Safe Place?" In *Chaos Communication Congress*, <https://lb1.eff.org/files/ccc2010.pdf>, 2010 (cit. on p. 203).
- [225] R. Holz, L. Braun, N. Kammenhuber, and G. Carle, "The SSL Landscape: A Thorough Analysis of the X.509 PKI Using Active and Passive Measurements," in *ACM SIGCOMM Conference on Internet Measurement*, ACM, 2011, pp. 427–444 (cit. on p. 203).
- [226] I. Ristic and M. Small, "A Study of What Really Breaks SSL," *Hack in the Box*, vol. http://blog.ivanristic.com/Qualys_SSL_Labs-A_Study_of_Really_Breaks_SSL-HITB_Amsterdam_2011.pdf, May 2011 (cit. on p. 204).
- [227] Z. Durumeric, J. Kasten, M. Bailey, and J. A. Halderman, "Analysis of the HTTPS Certificate Ecosystem," in *2013 Conference on Internet Measurement*, ACM, 2013, pp. 291–304 (cit. on p. 204).
- [228] F. Giesen, F. Kohlar, and D. Stebila, "On the Security of TLS Renegotiation," in *ACM Conference on Computer & Communications Security*, ACM, 2013, pp. 387–398 (cit. on p. 204).
- [229] P. Szalachowski, S. Matsumoto, and A. Perrig, "PoliCert: Secure and Flexible TLS Certificate Management," in *ACM SIGSAC Conference on Computer and Communications Security*, ACM, 2014, pp. 406–417 (cit. on p. 204).
- [230] M. D. Ryan, "Enhanced Certificate Transparency and End-to-end Encrypted Mail," *Proceedings of NDSS 2014*, The Internet Society, 2014 (cit. on p. 204).
- [231] J. Clark and P. C. van Oorschot, "SoK: SSL and HTTPS: Revisiting Past Challenges and Evaluating Certificate Trust Model Enhancements," in *IEEE Symposium on Security and Privacy*, IEEE, 2013, pp. 511–525 (cit. on p. 204).
- [232] L. S. Huang, A. Rice, E. Ellingsen, and C. Jackson, "Analyzing Forged SSL Certificates in the Wild," in *IEEE Symposium on Security and Privacy*, IEEE, 2014, pp. 83–97 (cit. on p. 204).
- [233] C. Brubaker, S. Jana, B. Ray, S. Khurshid, and V. Shmatikov, "Using Frankencerts for Automated Adversarial Testing of Certificate Validation in SSL/TLS Implementations," in *IEEE Symposium on Security and Privacy*, IEEE, 2014, pp. 114–129 (cit. on p. 204).
- [234] A. Bates, J. Pletcher, T. Nichols, B. Hollembaek, D. Tian, K. R. Butler, and A. Alkhelaifi, "Securing SSL Certificate Verification through Dynamic Linking," in *ACM SIGSAC Conference on Computer and Communications Security*, ACM, 2014, pp. 394–405 (cit. on p. 204).

Bibliography

- [235] S. Fluhrer, I. Mantin, and A. Shamir, “Weaknesses in the Key Scheduling Algorithm of RC4,” in *Selected Areas in Cryptography*, Springer, 2001, pp. 1–24 (cit. on p. 207).
- [236] X. Wang and H. Yu, “How to Break MD5 and Other Hash Functions,” in *Advances in Cryptology—EUROCRYPT*, Springer, 2005, pp. 19–35 (cit. on p. 207).
- [237] A. Popov, “Prohibiting rc4 cipher suites,” *Computer Science*, vol. 2355, pp. 152–164, 2015 (cit. on p. 207).
- [238] W. Diffie, P. C. Van Oorschot, and M. J. Wiener, “Authentication and Authenticated Key Exchanges,” *Design, Codes and Cryptography*, vol. 2, no. 2, pp. 107–125, 1992 (cit. on p. 207).
- [239] L. Catuogno and I. Visconti, “An Architecture for Kernel-Level Verification of Executables at Run Time,” *Computer Journal*, vol. 47, no. 5, pp. 511–526, 117 2004 (cit. on pp. 217, 219).
- [240] M. McKusick and G. Neville-Neil, *The Design and Implementation of the FreeBSD Operating System*. Addison-Wesley Publishing Company, Reading, MA, 2005 (cit. on p. 218).
- [241] G. Kroah-Hartman, “Signed Kernel Modules,” *Linux Journal*, pp. 301–308, 117 2004 (cit. on p. 219).
- [242] S. T. King, P. M. Chen, Y.-m. Wang, C. Verbowski, H. J. Wang, and J. R. Lorch, “Subvirt: Implementing Malware with Virtual Machines,” in *In IEEE Symposium on Security and Privacy*, IEEE, 2006, pp. 314–327 (cit. on p. 219).
- [243] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh, “Terra: a Virtual Machine-based Platform for Trusted Computing,” *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 193–206, 2003 (cit. on pp. 219, 220).
- [244] T. Garfinkel and M. Rosenblum, “A Virtual Machine Introspection Based Architecture for Intrusion Detection,” in *In Proceedings of the Network and Distributed Systems Security Symposium (NDSS 2003)*, 2003, pp. 191–206 (cit. on pp. 219, 220, 231).
- [245] J. Oberheide, E. Cooke, and F. Jahanian, “CloudAV: N-Version Antivirus in the Network Cloud,” in *Proceedings of the 17th USENIX Security Symposium*, San Jose, USA, 2008 (cit. on p. 220).
- [246] W. Yan and E. Wu, “Toward Automatic Discovery of Malware Signature for Anti-Virus Cloud Computing,” *Advanced Threats Research Trend Micro Inc.*, 2009 (cit. on p. 220).
- [247] M. Laureano, C. Maziero, and E. Jamhour, “Intrusion Detection in Virtual Machine Environments,” in *Proceedings of the 30th Euromicro Conference*, IEEE, 2004, pp. 520–525 (cit. on p. 220).
- [248] G. Wagener, R. State, and A. Dulaunoy, “Malware Behaviour Analysis,” *Journal in Computer Virology*, vol. 4, pp. 279–287, 2008 (cit. on p. 223).
- [249] A. P. Sheth and J. A. Larson, “Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases,” *ACM Computing Surveys (CSUR 1990)*, vol. 22, no. 3, pp. 183–236, 1990 (cit. on p. 232).

- [250] I. Botan, Y. Cho, R. Derakhshan, N. Dindar, A. Gupta, L. Haas, K. Kim, C. Lee, G. Mundada, M.-C. Shan, N. Tatbul, Y. Yan, B. Yun, and J. Zhang, "A Demonstration of the MaxStream Federated Stream Processing System," in *IEEE 26th International Conference on Data Engineering (ICDE 2010)*, IEEE, 2010, pp. 1093–1096 (cit. on p. 232).
- [251] Y. H. Harold Lim and S. Babu, "How to Fit when No One Size Fits," in *Proceedings of the Biennial Conference on Innovative Data Systems Research (CIDR 2013)*, 2013 (cit. on p. 232).
- [252] V. Gulisano, R. Jimenez-Peris, M. Patino-Martinez, C. Soriente, and P. Valduriez, "StreamCloud: An Elastic and Scalable Data Streaming System," *IEEE Transactions on Parallel and Distributed Systems (TPDS 2012)*, vol. 23, no. 12, pp. 2351–2365, 2012 (cit. on p. 233).
- [253] M. Sadoghi and H.-A. Jacobsen, "Analysis and Optimization for Boolean Expression Indexing," *ACM Transactions on Database Systems (TODS 2013)*, vol. 38, no. 2, 8:1–8:47, 2013 (cit. on pp. 233, 245).
- [254] M. K. Aguilera, R. E. Strom, D. C. Sturman, M. Astley, and T. D. Chandra, "Matching Events in a Content-based Subscription System," in *Proceedings of the Symposium on Principles of Distributed Computing*, ACM, 1999, pp. 53–61 (cit. on p. 233).
- [255] S. E. Whang, H. Garcia-Molina, C. Brower, J. Shanmugasundaram, S. Vassilvitskii, E. Vee, and R. Yerneni, "Indexing Boolean Expressions," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 37–48, 2009 (cit. on p. 233).
- [256] A. Forget, S. Komanduri, A. Acquisti, N. Christin, L. F. Cranor, and R. Telang, "Building the Security Behavior Observatory: An Infrastructure for Long-term Monitoring of Client Machines," in *Proceedings of the Symposium and Bootcamp on the Science of Security (HotSOS 2014)*, Raleigh, USA: ACM, 2014, 24:1–24:2 (cit. on p. 234).
- [257] J. Gionta, A. Azab, W. Enck, P. Ning, and X. Zhang, "DACSA: A Decoupled Architecture for Cloud Security Analysis," in *7th Workshop on Cyber Security Experimentation and Test (CSET 2014)*, 2014 (cit. on p. 234).
- [258] D. Srinivasan, Z. Wang, X. Jiang, and D. Xu, "Process Out-grafting: An Efficient "out-of-VM" Approach for Fine-grained Process Execution Monitoring," in *Proceedings of the Conference on Computer and Communications Security (CCS 2011)*, ACM, 2011, pp. 363–374 (cit. on p. 234).
- [259] L. Golab, T. Johnson, J. S. Seidel, and V. Shkapenyuk, "Stream Warehousing with DataDepot," in *Proceedings of the International Conference on Management of data (SIGMOD 2009)*, ACM, 2009, pp. 847–854 (cit. on p. 234).
- [260] T. Johnson and V. Shkapenyuk, "Data Stream Warehousing in Tidalrace," in *Proceedings of the Biennial Conference on Innovative Data Systems Research (CIDR 2015)*, 2015 (cit. on p. 234).
- [261] Z. Cao, S. Chen, F. Li, M. Wang, and X. S. Wang, "LogKV: Exploiting Key-Value Stores for Log Processing," in *Proceedings of the Biennial Conference on Innovative Data Systems Research (CIDR 2013)*, 2013 (cit. on p. 234).

Bibliography

- [262] L. Deri, S. Mainardi, and F. Fusco, "Tsdb: A Compressed Database for Time Series," in *Traffic Monitoring and Analysis*, vol. 7189, 2012, pp. 143–156 (cit. on p. [235](#)).
- [263] H. T. Vo, S. Wang, D. Agrawal, G. Chen, and B. C. Ooi, "LogBase: A Scalable Log-structured Database System in the Cloud," *Proceedings of the VLDB Endowment*, vol. 5, no. 10, pp. 1004–1015, 2012 (cit. on p. [235](#)).
- [264] S. Wang, D. Maier, and B. C. Ooi, "Lightweight Indexing of Observational Data in Log-Structured Storage," in *Proceedings of the VLDB Endowment*, 2014, pp. 529–540 (cit. on pp. [235](#), [249](#)).
- [265] A. Ailamaki, D. J. DeWitt, M. D. Hill, and M. Skounakis, "Weaving Relations for Cache Performance," in *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, 2001, pp. 169–180 (cit. on p. [249](#)).
- [266] M. A. Bender, M. Farach-Colton, J. T. Fineman, Y. R. Fogel, B. C. Kuszmaul, and J. Nelson, "Cache-oblivious Streaming B-trees," in *Proceedings of the Symposium on Parallel Algorithms and Architectures*, ACM, 2007, pp. 81–92 (cit. on p. [249](#)).

Curriculum Vitae

Diese Seite enthält persönliche Daten. Sie ist deshalb nicht Bestandteil der Online-Veröffentlichung.