Department of Computer Science

A Secure Peer-to-Peer Application Framework

Joakim Koskela

Aalto University

DOCTORAL DISSERTATIONS

Aalto University publication series
DOCTORAL DISSERTATIONS 14/2015

A Secure Peer-to-Peer Application Framework

Joakim Koskela

A doctoral dissertation completed for the degree of Doctor of Science (Technology) to be defended, with the permission of the Aalto University School of Science, at a public examination held at the lecture hall T1 of the school on 2 February 2015 at 12.

Aalto University
School of Science
Department of Computer Science

Supervising professor
Professor Jukka K. Nurminen

Thesis advisor
Professor Andrei Gurtov

Preliminary examiners
Professor George C. Polyzos, Athens University of Economics and Business, Greece
Professor Róbert Szabó, Budapest University of Technology and Economics, Hungary

Opponent
Professor Klaus Wehrle, RWTH Aachen University, Germany

Aalto University publication series
DOCTORAL DISSERTATIONS 14/2015

© Joakim Koskela

ISBN 978-952-60-6063-7 (printed)
ISBN 978-952-60-6064-4 (pdf)
ISSN-L 1799-4934
ISSN 1799-4934 (printed)
ISSN 1799-4942 (pdf)
http://urn.fi/URN:ISBN:978-952-60-6064-4

Unigrafia Oy
Helsinki 2015

Finland

NORDIC ECOLABEL
441 697
Printed matter

Aalto University

Abstract

Aalto University, P.O. Box 11000, FI-00076 Aalto www.aalto.fi

Author	Joakim Koskela
Name of the doctoral dissertation	A Secure Peer-to-Peer Application Framework
Publisher	School of Science
Unit	Department of Computer Science
Series	Aalto University publication series DOCTORAL DISSERTATIONS 14/2015
Field of research	Network architectures and security
Manuscript submitted	25 February 2011
Date of the defence	2 February 2015
Permission to publish granted (date)	18 November 2014
Language	English

[✓ **Monograph**] ☐ **Article dissertation (summary + original articles)**

Abstract

The nature of the Internet has changed dramatically. From a modest research network, it has evolved into one of the most important fabrics of our modern society, affecting the lives of billions each day. We rely on it for everything from performing our daily chores to accessing rich media and keeping in touch with our friends.

Despite this change, service provisioning has largely remained intact. Services are provided in a centralized manner, resulting in bottlenecks and vulnerable collections of, often unwittingly, submitted sensitive information. Peer-to-peer (P2P) technologies have the potential to provide a better alternative for future networking. P2P services distribute the load from a single node to a network of peers, relying on the resources of the end-users themselves. Not only does it remove the bottlenecks, it has the potential to provide a more personal and safe networking environment.

In this dissertation, we inspect the feasibility and implications of a generic, cross-application, P2P framework. We present the design and implementation of a framework that uses existing infrastructure and advanced networking protocols to create a secure environment. Using this framework, applications are able to benefit from P2P networking without having to deploy new infrastructure or implement complex connection- and identity management. Users benefit from using a single, strong, cross-application identity management and having better control over their data. This improves the trust within the system and enables new ways of dealing with security threats.

We demonstrate the feasibility of the framework by evaluating the performance and usability of the prototype implementation. This provides a model for future networking applications and insight into the security and usability issues these will face.

Keywords	peer-to-peer, security, mobility, identity management, application frameworks				
ISBN (printed)	978-952-60-6063-7	**ISBN (pdf)**	978-952-60-6064-4		
ISSN-L	1799-4934	**ISSN (printed)**	1799-4934	**ISSN (pdf)**	1799-4942
Location of publisher	Helsinki	**Location of printing**	Helsinki	**Year**	2015
Pages	198	**urn**	http://urn.fi/URN:ISBN:978-952-60-6064-4		

Preface

This thesis is the result of my work at Helsinki Institute of Information Technology (HIIT) during 2007-2013. First and foremost, I like to thank the institution and its staff, especially my foreman and instructor Andrei Gurtov, for supporting me during this time. It has been an inspiring environment to work in, and without the great colleagues, interesting conversations and opportunities to connect with researchers all over the world, this dissertation would not have been completed. Furthermore, I owe my gratitude to my supervisor Jukka Nurminen for encouragement and patience during the final phases of the process. I am also grateful to my former supervisor Antti Ylä-Jääski for the assistance in the beginning, and Zhonghong Ou for providing valuable feedback on early drafts of this dissertation.

I wish to thank my colleagues and co-authors Juho Heikkilä, Kristiina Karvonen, Dmitry Korzun, Sasu Tarkoma, Theofanis Kilinkaridis, Nicholas Weaver, Mark Allman, Jani Hautakorpi and Gonzalo Camarillo for the cooperation during these years, helping me refine ideas, as well as my writing. In addition, I have had the pleasure to work with several brilliant people, whose input has been invaluable. I would like to thank Miika Komu, Samu Varjonen, Ramya Sri Kalyanaraman, Andrey Khurri, Boris Nechaev, Dmitriy Kupsov, Tatiana Polishchuk, Andrey Lukyanenko, Ilya Nikolaevskiy, Oleg Ponomarev, Ruishan Zhang, Cheevarat Jampathom and Ajit Kumar for being great colleagues, and friends.

The work was completed as part of various projects at HIIT, including Trustworthy Internet (TrustInet), Infrastructure for HIP (InfraHIP), Secure Peer-to-Peer Services Overlay Architecture (SPEAR), NordicHIP for which the Future Internet Graduate School (FIGS), FICNIA, TEKES, the Academy of Finland, the NLnet foundation and funding partners deserve recognition for enabling.

I would also like to express my gratitude to the research institutions and groups I have collaborated with during these years. My visit at the International Computer Science Institute (ICSI) in Berkeley, California influenced my approach to research greatly, and I will always be grateful for the warm welcome and support I received. I am glad to have met, and visited, the people from the Distributed Systems Group at RWTH Aachen University, especially Tobias Heer, René Hummen and Stefan Götz, who provided interesting conversations and good company. Finally, I would like to thank Miika Tuisku and Tommo Reti for giving me the opportunity to work with the Helsinki Institute of Physics Technology Programme (HIP-TEK) at CERN, Geneva, which provided an insight into the commercial aspects of distributed computing, as well as fresh ideas.

I want to thank the pre-examiners of this dissertation, professor George C. Polyzos of the Athens University of Economics and Business, and professor Róbert Szabó of Budapest University of Technology and Economics, and professor Klaus Wehrle of RWTH Aachen University for agreeing to serve as my opponent.

Finally, I thank my family and friends for the support, love and guidance I have always received. Especially Ulli for the love and compassion you have shown, I look forward towards sharing the joys and challenges of the exciting times to come.

Helsinki, January 13, 2015,

Joakim Koskela

Contents

Author's Contribution

The work for this dissertation started at the TrustInet [1] project with the intent of studying how trust-based solutions can be used to enhance security in distributed environments. The project begun with simple experiments on call filtering using a P2P SIP prototype. It quickly grew into a generic platform for evaluating security mechanisms in distributed applications. As of today, the platform has been used for experimenting with a number of security mechanisms and different applications.

The main contributions of this dissertation are:

- *The design and evaluation of a cost-effective secure P2P application framework.* Using a prototype for mobile devices, we evaluate our P2P application framework that enables applications to securely communicate in distributed environments. The feasibility of our framework is demonstrated using performance measurements, protocol analysis and usability evaluation. Using infrastructure based on redundancy of independent unreliable resources, we show that networks can be built cost-effectively, and quantify their limitations. Through these evaluations, we show that our framework provides a less expensive, but secure, alternative for building generic overlay networks.

- *A secure model for P2P web.* We design and evaluate P2P web integration using a novel security scheme and a distributed caching model. Combined with the security mechanisms of our framework, it provides protection both from eavesdropping and malicious content. Unlike traditional approaches to distributed caching, we do not assume all nodes to be trustworthy, but include the security needed to provide authenticity and protect the privacy of users. The model is evaluated using a

[1]Trustworthy Internet (TrustInet) http://trustinet.hiit.fi

prototype implementation.

- *A privacy-enhancement scheme for identity-based P2P networks.* We analyze the privacy issues of strong identity-based P2P networks and develop a generic scheme that allows users to hide their actions from members of the overlay, while still leveraging resources shared by those. The scheme is implemented and evaluated, with special focus on the performance and usability of the solution. To our knowledge, this is the first complete attempt at solving privacy issues in P2P networks based on persistent identities, as traditional approaches concentrate only on hiding the current owner of a dynamic identity.

- *A privacy-conscious mechanism for traffic filtering in P2P environments based on relationships.* We developed a Bloom filter- based solution for virally sharing relationship information, suited for traffic filtering. We analyze the security issues of viral relationship sharing, and present a novel protocol for the safe, and privacy-preserving, distribution of these. The scheme is evaluated with metrics recorded from from real-life social networks, and compared to current solutions with regards to the security of the solution and overhead. We show that it efficiently protects both sides of an exchange, and provides, with a high degree of certainty, a relationship path between the two while keeping the intermediate links anonymous.

This dissertation does not address the specifics of any particular P2P overlay protocol. Topics such as churn, routing overhead and network-level scalability are not discussed as far as they concern the overlay structure. We concentrate on the structure and security of applications using these overlays, and leave the networking details to the designers of those protocols.

The design of the framework was done jointly with Juho Heikkilä and Andrei Gurtov, although the author acted as the main architect and was responsible for implementing, and evaluating, the prototype. The design was published in [64] and [98]. In [64], the author contributed to the architecture description and was responsible for developing a method for efficiently distributing the service discovery process. The identity model, and use of mobile cross-application identities, is a topic the author brought to the framework from his related research [104]. The author was responsible for designing a generic interface which allowed the integration of external identity management systems, as well as to integrate the system used in [104].

Application support, web integration and cooperative caching, as described in [98][99], was designed and implemented by the author. The author was also responsible for designing and executing the feasibility evaluations, as well as developing the optimizations.

The P2P web security enhancements are based on work done jointly with Nicholas Weaver, Mark Allman and Andrei Gurtov on adding attribution and integrity to web pages using cross-application identities [104]. The author was responsible for developing the concept, and conceiving a way of integrating it into web pages and servers, as well as the development of the prototype system. The author was also responsible for adapting the concept to P2P environments, and integrating it with the P2P framework.

The privacy enhancements were conceived and designed by the author. The prototype implementation and functional evaluation was completed jointly by the author and Cheevarat Jampathom. The results were published in a conference paper [103] written jointly with Professor Sasu Tarkoma of the University of Helsinki.

The study of the usability aspects of the security solutions, and VoIP in general, was done by Theofanis Kilinkaridis and Kristiina Karvonen. The author was responsible for providing technical details, and for defining the bounds for the paper mock-ups. The initial user interface for the security enhancements was designed by the author, although influenced by discussions with Theofanis Kilinkaridis and Kristiina Karvonen. The work of evaluating the interface, and subsequent improvements, were done by Kristiina Karvonen, Sanna Shibasaki and the author. Parts of this work has been published in [102] and [101].

Filtering traffic based on social networks (relationships) is a topic conceived by the TrustInet project group and developed by Juho Heikkilä. The author's role was to design an alternative mechanism suitable for fully P2P networks. The author was responsible for studying how Bloom filters could be used in a secure, and privacy preserving manner, and conceiving and implementing the solutions presented in this thesis. This work was has been published in [100].

List of Figures

List of Tables

List of Abbreviations

AOR	Address of Record
API	Application Programming Interface
BEET	Bound End-to-End Tunnel
BEX	Base Exchange
CPU	Central Processing Unit
DHCP	Dynamic Host Configuration Protocol
DHT	Distributed Hash Table
DNS	Domain Name System
DSL	Digital Subscriber Line
ESP	Encapsulated Security Payload
BSD	Berkeley Software Distribution
GB	Giga Byte
GHz	Giga Hertz
GNU	Gnu's Not Unix
GUI	Graphical User Interface
HI	Host Identity
HIIT	Helsinki Institute for Information Technology
HIP	Host Identity Protocol
HIPBONE	HIP-Based Overlay Networking Environment
HIPL	HIP for Linux
HIT	Host Identity Tag
HTTP	HyperText Transfer Protocol
HTTPD	HTTP Daemon
HTTPS	HTTP Secure
ICE	Interactive Connection Establishment
ICMP	Internet Control Message Protocol
IEEE	Institute of Electrical and Electronics Engineers
IETF	The Internet Engineering Task Force

IM	Instant Messaging
IP	Internet Protocol
IPSec	IP Security Architecture
ISP	Internet Service Provider
ITU	International Telecommunication Union
KB	Kilo Byte
KDE	K Desktop Environment
LAN	Local Area Network
MAC	Media Access Control
MB	Mega Byte
MD5	Message-Digest algorithm 5
MTU	Maximum Transmission Unit
NAT	Network Address Translation
OS	Operating System
P2P	Peer-to-Peer
P2PSIP	Peer-to-Peer SIP
PC	Personal Computer
PGP	Pretty Good Privacy
PHP	PHP: Hypertext Preprocessor
PKI	Public Key Infrastructure
POTS	Plain Old Telephone System
RAM	Random Access Memory
RFC	Request For Comments
RPC	Remote Procedure Call
RSA	Rivest, Shamir and Adleman
RSS	Really Simple Syndication
RTT	Round Trip Time
RVS	Rendezvous Server
SHA	Secure Hash Algorithm
SIP	Session Initiation Protocol
SOAP	Simple Object Access Protocol
SPIT	Spam over Internet Telephony
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UA	User Agent
UDDI	Universal Description Discovery and Integration
UDP	User Datagram Protocol
UI	User Interface

URL	Uniform Resource Locator
USB	Universal Serial Bus
VoIP	Voice over IP
WLAN	Wireless LAN
WOT	Web of Trust
WSDL	Web Services Description Language
WWW	World Wide Web
XML	eXtensible Markup Language

1. Introduction

Peer-to-peer (P2P) systems have been a popular research topic during the past years, as they have the potential to offer more reliable, fault-tolerant and cost-efficient networking. As these systems are built on the resources of the participating client nodes themselves, they are not dwarfed by the reliability, trustworthiness and performance of dedicated 'bottleneck' servers. The advantages are well known, and over the past decade, we have seen a surge of applications which to some degree rely on P2P technologies.

Although the concept is as old as computer networking, service provisioning has long followed a *centralized* model. A widespread, serious, interest in P2P service provisioning did not begin before the end of the last century when file-sharing systems such as Napster [1], the Gnutella network, and more recently BitTorrent and the Skype[2] messaging application, managed to demonstrate the enormous potential of P2P technologies in providing scalable solutions to otherwise infeasible services.

Meanwhile, the nature of the Internet has evolved from an encyclopedic-like directory of information to a social medium, where people interact and express themselves. Much of the content we see on the Internet today, such as photographs, blog posts and product reviews, is produced by the users themselves. Even sites without user generated content often embed elements that enable social interaction, such as *like* and *share*- buttons that allow users to recommend content to others. However, the end-user has no real control over how this information is used. The billions of bits of data produced each day are stored in the service providers' databases, creating vast collections of sensitive information.

In addition to being more scalable, P2P technologies have the potential

[1]http://www.napster.com
[2]http://www.skype.com

to provide a safer networking environment for this modern, social, Internet. As the users themselves take part in the provisioning, they have better control over how information is managed. However, P2P- based services are still few in today's Internet, and considered special, technically complex. Each one has required carefully designed solutions for connectivity issues and security threats, that fit the characteristics and environment of that particular application.

This has lead to highly specialized solutions optimized for a single cause, and to the deployment of specifically tailored infrastructure. The service model is decided early on, and the application is built very aware of the network topology, whether P2P or centralized, it operates in. Changing the underlying service model is costly, requiring a fundamental reevaluation of how the application works, and many known, and unknown, issues to be solved. Furthermore, as P2P implies that a provider relinquishes some of the control over the service, investing in P2P is commitment that is hard to justify unless there are immediate benefits, such as when the application is exceptionally resource intensive.

The topic of this dissertation is to address the application-level issues that raise the threshold for utilizing distributed service provisioning. By designing a generic P2P application framework, we have created an environment that allows us to study the requirements of these applications, and to quantify and develop solutions to the problems encountered. However, our focus is not on solely P2P-specific issues, but rather on how applications should be designed, and what it would mean for them, to be oblivious to the underlying network structure. This would enable services that can be hosted either way, thus easing the transition for both existing, as well as completely new, types of applications to special environments.

Compared to existing research on P2P networking, our focus is not on the structure and protocols of the overlay network, but rather on how applications, even existing ones, adapt to environments without a fixed structure or trusted authority. We sketch an overall architecture within which different, current or future, network protocols can be used. Secondly, we do not assume a specific type of networking application, but examine the issues common to all, independent of their data transfer patterns or other characteristics. We do not assume that the applications are even aware of the underlying framework, but create a system which can be combined even with unmodified legacy applications.

Much of our work revolves around the synergy and security of applications, i.e., how a dispersed set of users can bootstrap new applications and communicate safely without a centralized authority. But we also examine more practical issues, such as ensuring availability and fairness. Creating a P2P framework that could explore these issues has conventionally been a daunting task, requiring commitment and investment to both the framework and the applications meant to use it. Fueled by the recent explosion of consumer-oriented cloud services, our insight is that not only can we utilize existing, readily available infrastructure and protocols to provide such a framework, but we can adapt existing applications to it.

As the work presented in this dissertation is based on experiments using our generic P2P framework for mobile devices, we begin by reviewing the design, placing special emphasis on the strong identity scheme which is the cornerstone of the system's security. Through evaluations and measurements, we show that P2P overlays can be deployed at low cost using readily available generic *cloud* resources, while remaining secure and responsive.

We continue by describing the application-level issues we have studied in this environment, and presenting our solutions. First we study the privacy issues related to using the unreliable peer resources in P2P networks. We develop a simple scheme, and discuss the usability implications of it. Secondly we address relationships and trust in distributed systems using a novel privacy- conscious introduction mechanism. We continue by examining data availability and fairness in these networks by studying a distributed web caching scheme and presenting the improvements to a novel incentives mechanism for data distribution. We finish with our findings from a study of the usability of distributed communication systems, presenting our solutions to making security usable in these environments.

1.1 Problem statement

The goal of the research of this dissertation is to develop and explore different aspects of a secure P2P framework. We believe that P2P provides a more natural fit for future networking applications, as the focus is increasingly shifting towards mobile access and user- centric interaction. However, creating new applications for these environments is difficult. There are a number of issues to consider, including deployment, security and usability, to which no obvious solutions exist. These are the problems we are addressing. Specifically, the main research problems are stated as follows.

i How do we design a P2P framework that is easily deployable in both present and future networking environments?

ii How do we manage identities and security without a centralized authority?

iii How do we design a generic framework that supports a wide range of applications and networking models?

As for the first question, deployability is a key issue when designing new software. Our goal is not to design the most efficient network protocol, but rather to create a framework that is practical and can actually be deployed in the current, and future, Internet. An efficient way of achieving good deployability is to re-use existing technologies and resources in an innovative manner, which is the approach we have taken. To better adapt to future networks, we set out with few requirements or assumptions regarding the intended network environments. By abstracting the resource management based on the least common denominator of what is available today, our goal is to develop a robust framework which not only is deployable in various networks, but also migrates easily between those.

Aiming at producing a *truly* P2P framework, suitable even for mobile and ad-hoc networks, we need a design that replaces the traditional centralized approach to identity management with a distributed solution. The integrity of identities is a crucial prerequisite for any type of security, however, it is not the only issue related to identity management in distributed environments. Privacy and how peers relate to each other, the *trust* between users, must also be carefully considered.

As P2P systems form a network between the participating nodes, used to collectively perform tasks and manage resources, the network itself becomes the service provider. Where there once was a trusted, centralized component, we now have a collection of possibly non-trusted nodes. Most P2P systems have few mechanisms in place for controlling how information is propagated within the network. Considering a P2P telephony application, signaling exchange made through the network (such as media content, call set-up, messaging or presence information) can potentially be intercepted and abused, or forged, by any one of the peers. This raises concerns about privacy and how to protect information, as well as places the dependability of the nodes in the spotlight.

Without a trusted provider to govern users and protect us from malicious peers, we have to rethink how we establish and use trust in these environments. We can only rely on what we know ourselves, our *experiences*, as well as what we have heard from others we trust. Furthermore, as computer networks spread across borders and cultures, even the notion of what is correct, or trustworthy, will differ. We should explore how we can design a model that uses our own knowledge and assumptions to judge what is safe, while providing a sufficient level of security.

Also, as we are designing a P2P *framework*, we should consider identity and content management broadly, including how different applications could benefit from having unified framework-managed identities. We should consider what it means to have users represented in a unified manner across different applications. It may create new opportunities and ways of communicating, but can also carry unwanted side effects. We should rethink what content and information sharing in such an environment means. By using strong identities and different types of networking, we transform on-line sharing from being tied to a specific service provider to an activity where the service providers and networks we choose act merely as a medium for communication.

Finally, as the purpose of the framework is to serve a range of applications instead of only a few (as most existing P2P systems), one of the research problems is to examine and design ways in which applications can interact with it. In addition to new application programming interfaces (APIs), this includes support for existing legacy applications such as voice and video-call applications. We want to create a framework which can transform these from a centralized model into P2P applications suitable for private, mobile or ad-hoc networks. This requires careful planning of the application interfaces the framework provides, and other integration details.

1.2 Methodology and tools

Large parts of the research is based on experiments conducted using an implementation of a P2P application framework prototype. The approach has therefore been a combination of background research, empirical evaluation, experimental analysis and engineering effort.

The process for researching the topics of this dissertation is best described as an *iterative* process. After a research problem arose, the topic was analyzed, identifying possible sub-components and the areas of the application framework it affected. Literature surveys were conducted throughout the process, to identify possible solutions and gain a deeper understanding of the issues. The first steps towards a solution to the problems were taken using qualitative analysis. The factors affecting the problem, and possible solution models were sketched. This was followed by initial empirical analysis of the proposed solutions, where prototypes and simulations were implemented and experimented with using either real-world, or constructed, data sets. Following the initial validation, a prototype for the framework was implemented and evaluated, usually using the Python plug-in interface of the framework. The solution was further optimized and improved based on the results, or discarded if it was found faulty or infeasible. Finally the implementation was finalized, and deployed.

Much of the experimental results were obtained from a test-bed set up at the Helsinki Institute for Information Technology (HIIT) office in Espoo, Finland. For certain experiments that required data sets not feasible to collect as part of this dissertation, models from well-known sources (such as real-world social networks) were used.

The primary platform for the research was the Nokia Internet tablet N810. Although these are not considered to be a high-end mobile devices today, feature-wise they represent what we expect from a modern, handheld, computing environment. The N810 was selected due to its good availability at the start of the research, as well as for its excellent development environment. The framework was developed using the Meamo SDK[3], which relies on the Scratchbox[4] cross-compilation toolkit. The software engineering was done using desktop- and laptop computers, using the Vim[5] (Vi iMproved) and GNU Emacs[6] editors and the GNU toolchain and compiler collection[7].

The early prototypes, as well as most server-side software, were built using the Python[8] interpreter. Measurements were conducted using a range of tools, including *iperf*, *ping*, *wget* and *curl* on Debian-based desktop computers. Hip for Linux (HIPL) version 1.0.4 was used to provide HIP support, and Miredo 1.1.5 for Teredo networking. A set of bash shell

[3]http://maemo.org/development/
[4]http://www.scratchbox.org
[5]http://www.vim.org
[6]http://www.gnu.org/emacs/
[7]http://gcc.gnu.org
[8]http://www.python.org

scripts, as well as LibreOffice[9] was used to summarize the data and construct charts.

1.3 Structure of Dissertation

The dissertation is organized as follows. Background on the relevant technologies and existing systems which have influenced our work, or show similarities, is given in Chapter 2. Following the background, we present the rational behind, the design and finally the implementation of our solution in Chapter 3. We explain the different assumptions and requirements that affected the design, and how these were met. We continue by describing the key design principles and how these are applied to current and future networking environments, complete with an overview of the prototype implementation. In Chapter 4 we review the implementation, providing results from our evaluation of it. Through network performance measurements, observations from use on mobile devices and static analysis of the protocols we assess the feasibility of our approach.

This is followed by chapters discussing in detail different security enhancements and other features we have studied using the framework. Chapter 5 explores privacy issues in distributed environments, and describes the design and implementation of a privacy-enhancing mode for our framework.

Chapter 6 addresses two fundamental challenges of P2P Web: security and accessibility. We present a novel method for adding accountability to the web, which combined with a cooperative caching scheme enables a secure and usable Web experience in distributed environments. Chapter 7 describes our work with relationship-based traffic filtering, where we present a novel Bloom filter- based scheme for distributing relationship information in a privacy- conscious and secure manner.

Finally, we directed our attention to the usability of our solution, and P2P communication in general. In Chapter 8 we study attitudes and assumptions towards mobile P2P communication, and describe the user interface design process. We analyzed the usability of the current prototype and identified how the issues discovered could be improved. This dissertation ends with a discussion on the current status, together with future work in Chapter 9, and conclusions in Chapter 10.

[9]http://www.libreoffice.org

2. Background

This chapter presents an overview of the technologies and existing work relevant to our framework. We start by reviewing P2P concepts and technologies. We continue by introducing public-key cryptography and the concept of strong identities. We conclude by presenting relevant network communication protocols and reviewing security and connectivity issues seen in the Internet today.

2.1 Peer-to-peer technologies

The modern understanding of a P2P system is a distributed communication system where the end-users themselves contribute resources to sustain the system. By contrast, the current model for providing most of the network-based services is through a *centralized* approach. In these systems, the service provider operates all the resources (such as servers and networking capabilities) needed to maintain the service, while the users only connect to access those. Even though the service itself provides interaction and cooperation between users, on the networking-level users only interact with the service provider. The content shared (such as images or chat messages) is actually only shared with the service provider, who can then decide how it is used, or to whom it is made visible.

P2P systems allow users to network without service providers, by having the resources needed to maintain the system provided by the participating users themselves. Although P2P-like networking in different forms has been utilized since the dawn of the Internet, it has rarely been between the end-users themselves, but rather between different types of nodes within the network (distributed services). The modern type of P2P networking, where the end-users' devices actively participate to provide the service, was popularized in late 1990 by file-sharing systems. Figure 2.1 illustrates these different ways of providing networking services.

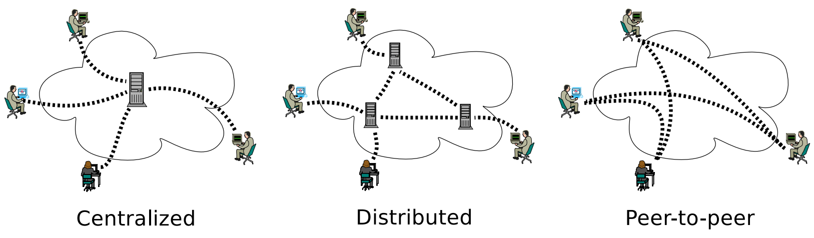


Figure 2.1. Different types of network services. Centralized services are provided by a (logically) single node, while distributed use a network of interconnected nodes. Peer-to-peer services utilize the resources of the end-users.

P2P networking is an active area of research, and over time we have seen numerous contributions helping the technology to evolve into being more reliable and robust. The first modern P2P systems were quite simple, often relying on a centralized resource lookup service, and using P2P connections only for data traffic. Modern systems incorporate efficient information lookup without a single point of failure, advanced network traversal techniques and anonymization of both content and users. In the following we review different classifications of P2P systems, including real-life examples of them. Furthermore, we summarize the different advantages and issues with using P2P technologies.

2.1.1 Peer-to-peer network structures

P2P systems can be divided into *structured* and *unstructured* based on the algorithm used to organize the network and propagate information within it. Structured P2P systems are based on organizing the network so that information is sought in a deterministic manner using a specific algorithm, such as a Distribute Hash Table (DHT). These systems are based on assigning identifiers to nodes and organizing the nodes so that data destined for a specific node identifier can be routed in an efficient manner. Data lookup is commonly based on assigning identifiers to the resources from the same name space as the node identifiers, and distributing the responsibility of these based on the proximity to the node identifiers.

Unstructured P2P systems are constructed in a more ad-hoc manner, where nodes join the network at random locations. Information propagation is usually relatively inefficient, often relying on flooding and time-to-live values. In many ways these mimic the natural patterns of information propagation; knowledge is shared with those that happen to be in the vicinity. Unstructured P2P networks are therefore better suited for casual information sharing, for systems where simplicity and proximity is appreciated. Structured P2P systems are designed for systems where performance and completeness is of key importance.

Unstructured P2P systems can further be divided into three subgroups based on the architecture of the network: Centralized, Pure and Hybrid P2P systems.

2.1.2 Network topologies

Centralized P2P systems are systems where a single centralized node is used for various tasks such as resource lookup (indexing) or for bootstrapping the system. The centralized node acts as a rendezvous for the peers, allowing them to locate each other, while the actual application data (e.g., a file transfer) is exchanged directly between the peers. Although the application data is exchanged without interference of the centralized node, these systems share many of the problems of centralized client-server systems such as single-point of failure and performance bottlenecks. However, these are also the easiest to create, manage and use. With centralized access control and authentication, the peers are relieved from much of the responsibility of keeping the system secure. This is also reflected in the usability of the applications, as they differ hardly at all from those based on centralized networking. In many ways, these can be seen as merely a variation of centralized systems where only a single task is delegated to P2P networking.

Pure P2P systems are systems where all peers contribute equally to the system. These systems do not suffer from having a single point of failure, but can be inefficient as queries may have to traverse a large number of peers as each single one has a very limited view of the network. These are also the most complex to set up and manage. System designers have to consider issues such as routing, security and usability carefully. Without a central authority, it is hard to identify and exclude malicious or misbehaving peers. Verifying that the data exchanged through the network is correct and remain unaltered while in transit requires a good security framework. Routing within the network also becomes complex as the peers responsible for forwarding packets may enter and leave at will.

Hybrid P2P systems try to combine the best features of both pure and centralized P2P systems. In hybrid networks, a subset of the peers (often referred to as super peers) is given more responsibility based on their resources or strategic position within the network. These do not suffer from having a single point of failure as centralized systems, while offering better performance than pure P2P systems. The super peers are used for rendezvous and indexing of resources, reducing the amount of peers requests need to traverse. Figure 2.2 depicts the different types of P2P network topologies.

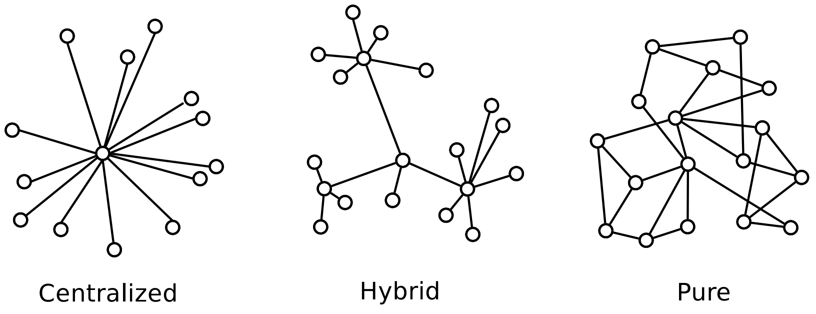


Figure 2.2. Illustration of the different types of P2P network topologies.

2.1.3 Peer-to-peer network examples

In the following, we review the details of two P2P protocols, one using a structured architecture and one using an unstructured one to illustrate the difference.

Gnutella

The Gnutella network is considered to be the first completely decentralized P2P network to gain a wide popularity. The Gnutella network was created for sharing files using keyword searches. Its primary task is to route user-made content queries and the corresponding responses, as well as to assist in creating the connections for the actual file transfer. The original Gnutella network consisted entirely of nodes of equal value that all participated in the maintenance of the network. When joining the network, a node established a pre-defined number of connections (originally five) to peers already within the network. The location of these nodes were found using, for instance, dynamic lists stored on well-known web locations.

When submitting a query, it was forwarded to each of the connected peers, who further forwarded it to their connections. Queries contained a hop-counter (time to live, TTL, counter), which was increased each time. This counter thus kept track of the spread of the query, and queries were dropped after traversing a specific distance (originally seven hops) to prevent the network from being filled with infinite queries.

The Gnutella network was in this manner fully distributed and put an equal amount of load on each peer. However, due to the flooding mechanism, the traffic within the network grows exponentially relative to the network size. This leads to the system easily being overloaded, and is therefore not considered *scalable*. To overcome these problems, and also enable lighter, resource-constrained clients, the Gnutella protocol has since evolved. Currently it features a hybrid architecture where *super nodes*, nodes with greater network and processing capacity, connect to a large number of hosts and serve as a proxy for regular clients. Furthermore, due to this new architecture, the maximum hop count of queries has been limited and experiments have been made for routing the query responses directly to the query initiator's super node instead of through the Gnutella network.

Chord

The Chord [167] protocol is one of the first structured P2P protocols developed. It provides an m-bit key-based lookup with an *O(*log*N)* node traversal guarantee (where *N* is the number of nodes in the system) for finding a piece of content in a P2P network. The size of the key space (value of m) is chosen according to the intended use of the system to provide a space sufficiently large to minimize the probability of overlapping key values. Applications of chord use these keys, in a hash table- manner, to retrieve pieces of data from the network, hence the term distributed hash table (DHT).

The Chord network is often illustrated as a ring where the participating nodes and content are distributed. Each node is assigned a position on the ring numbered 0 to 2^m (where m is the key space size). Chord relies on consistent hashing (such as SHA-1[43]) to assign the position (e.g., the hash value of the node's IP address) as well as for generating the content keys. The statistically unique and uniformly distributed properties of the hash function in use are integral for the Chord network to create a well-balanced network.

When joining a Chord network, the node is assigned its position (*peer identifier*) from the key space. With this assignment, it is given responsibility for maintaining all content which have a key value in the range between itself and the previous node on the Chord ring. Due to the properties of the key- and position-assigning algorithms, in a network with K keys and N nodes, each node is responsible for roughly K/N keys.

Each node maintains an up to m-entry table of the *successor* nodes (the

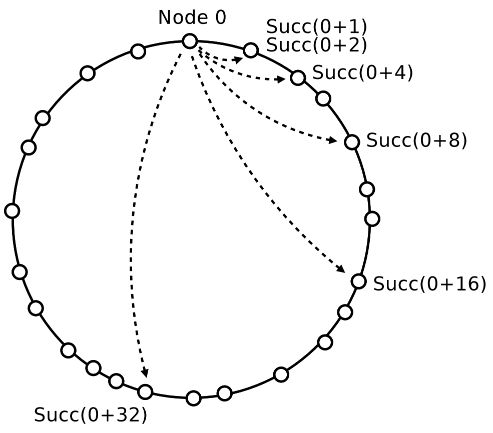


Figure 2.3. A 6-bit Chord ring illustrating the finger nodes of node 0.

closest node clockwise on the ring) for the values $n + 2^i$, where i ranges from 0 to m. When receiving a query for a key, unless the node itself is responsible for it, the query is forwarded using this table to the closest preceding node on the ring (which may forward it further). As these finger tables are logarithmic, each forward will bring the query closer to the responsible node by at least half of the remaining distance.

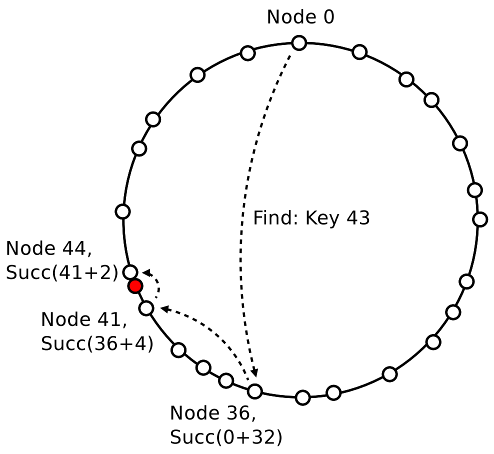


Figure 2.4. Illustration of the message propagation in Chord when fetching key 43.

Figures 2.3 and 2.4 illustrate the finger table and lookup process respectively in a six-bit Chord ring. Other structured P2P protocols include the Content-Addressable Network (CAN) [139], Tapestry [182], Pastry [147] and Kademlia [116] These represent the early work on structured P2P systems and DHTs, and are often the basis for current research. The key differences between these are found in how they manage the exchange of nodes in the network, routing logic and the amount of redundancy in connections and data placement. The exchange (joining and leaving) of nodes often result in a flood of maintenance traffic, which is important to optimize as it can overload the network in real-life scenarios. Redundancy and routing logic on the other hand affects the performance and stability of the networks.

2.1.4 Applications of peer-to-peer networking

As previously noted, P2P networking is not a novel concept, in fact, much of the Internet is built on the same concepts. Even among the first Internet applications to gain a wide popularity we find examples, such as email and Usenet news (Network News Transfer Protocol, NNTP), that are based on a decentralized model. Even though these were truly P2P systems, as the hosts participating in the network were also used by the end-users (even if through remote terminals), with the introduction of personal computing the participating nodes took on the role of service providers for the end-users' clients. It was not until it became common for end-users to handle multimedia and other large data that the need for P2P networking re-emerged.

With the surge of home Internet connectivity, the amount and ease of sharing data on a wider scale grew exponentially. However, sharing was very limited compared to current standards, often in private circles inaccessible to the public. Using P2P data transfers enabled public sharing systems to emerge, as it significantly decreased the load on the sharing sites. Today, file sharing remains the most popular use of P2P networking, with one application in particular, BitTorrent[1], responsible for a large share of all Internet traffic.

Another popular use for P2P is within personal communication applications. The best known example of these is Skype [2], a free communication system allowing voice and video telephony, as well as instant messaging and file sharing. Although the system is closed and uses proprietary technology, analysis has shown that it operates using a hybrid P2P network model [17]. This can be seen as a key enabler in its success, as a centralized model would have required large investments in infrastructure which might have made it impossible to offer the service for free. Other P2P communication systems and initiatives have since emerged, including Peer-to-Peer SIP (P2PSIP), presented in more detail in Chapter 2.5.

[1]http://www.bittorrent.com
[2]http://www.skype.com

Structured P2P networks are often used in systems where the performance and completeness can be guaranteed. Due to these characteristics, they can be used also for storing and retrieving data, rather than just as a fabric for pairing peers. Examples include distributed data storage and file systems [148] [36], caching [76] [175] and content search (such as the FAROO[3] and YaCy[4]). These systems have been shown to be robust and reliable even though a large portion of the overlay's nodes would fail [36]. Furthermore, structured P2P networks can also be used in conjunction with unstructured P2P systems. Especially centralized P2P systems can benefit from replacing the centralized components with a structured P2P network for increased robustness, as demonstrated by the Bittorrent DHT tracker.

2.1.5 Tradeoffs of peer-to-peer networking

P2P systems have several benefits over traditional client-server systems. As the peers contribute resources (network bandwidth, storage or computation power), the total capacity of the network increases as peers join, which makes it more scalable in theory. Except for centralized P2P systems, the network may also become more reliable and robust as there is no single point of failure. However, as P2P networks use the resources of the clients, weak clients can negatively impact the performance of the whole network. Furthermore, the additional load put on the clients may be problematic for some (such as mobile devices), where power consumption is critical and the storage capacity is limited.

However, the performance and efficiency of even the tinyest computing devices evolve over time, giving us more capacity and networking capability per Watt of energy. In recent studies, it has been shown that even for small hand-held mobile devices (such as PDAs and smartphones), P2P networking is by no means infeasible [125] [129] [126]. Furthermore, it has been shown that it is possible to modify, or *tune*, P2P systems into being better suited for mobile devices [91] [89] [90]. For instance, by paying attention to communication patterns and implementing smart idle sensing [60] and grouping of messages [14], or using probabilistic message dropping [92] substantial energy savings can be made while maintaining acceptable network performance. Another approach is to leverage the mobility of nodes, which has resulted in systems such as PReCinCt which

[3]http://www.faroo.com/
[4]http://yacy.net/

leverage proximity of the nodes for optimal performance [153].

Minimizing the negative impact weakly performing clients have on the network has been an important topic for P2P research. Unless the network is designed to consider the performance of the nodes in its structure, the network may easily become unresponsive. There are a number of reasons for clients to perform poorly in P2P networks. Resource-limited clients, such as mobile phones, simply do not have the computing power or storage capabilities of desktop computers. Furthermore, the network connection may be slow, congested or unreliable. Furthermore, the client may also *cheat* by showing poor performance even though having sufficient resources. For instance, network connectivity might be billed by the amount of data transferred. This, of course, forms a very strong incentive for the user to minimize his involvement in the P2P network. This problem of *free-riding* [8] can be mitigated in different ways; by minimizing the amount of these peers in the network, or their impact on the performance.

By providing the right incentives for users to share their resources, users are discouraged from free-riding, increasing the overall network performance and preventing even a total collapse [87]. These incentives can either be positive (rewarding the user when following the rules), or negative (punishing the user for performing poorly). The success of the Bittorrent protocol is, to a large extent, due to its incentive system. The Bittorrent protocol uses a so-called *tit-for-tat* mechanism for deciding how resources are distributed. This mechanism forces users to simultaneously share data of the files they are downloading in order to maintain a good download speed, leading to fairer sharing [34]. The Bittorrent network itself uses a centralized- or distributed P2P model, where *trackers* (either single nodes or DHTs) keep track of the files shared, and the users sharing them.

The success of Bittorrent has since led to extensive analysis of it [77] [135], resulting in mathematical modules for how different conditions and behavior affect its performance [136], as well as being seen as a standard to which new protocols are compared [155]. Furthermore, the tit-for-tat incentive mechanism used in Bittorrent has been questioned, leading to numerous improvements [20] [133] [109] [154]. Of course, Bittorrent is a relatively simple P2P application where the incentive mechanism is designed for rather short-sighted goals (maximum momentary throughput from immediately connected peers). For complex and long-lived systems, a more fine-tuned and widespread reputation system needs to be deployed

[47]. These need to be closely integrated with the identity management of the network to be efficient [48].

However, as there will always be peers that cannot affect their poor performance (due to resource constrictions), introducing incentives only to weed out the free-riders does not address the issue completely. By designing the P2P network in an intelligent manner which acknowledges the different restrictions of the individual nodes, we can avoid potential bottlenecks. By using different types of hybrid P2P network architectures, the traffic flow can be optimized. With smarter routing methods, the impact of congested and poorly performing peers can be minimized [172] [97]. And by using replication and redundancy in data storage, availability is increased [112].

Security is, however, harder to manage in P2P systems. As users rely on each other, there are usually ample opportunities for malicious activities. Impersonation, denial of service and distribution of bogus or malicious content are some of the problems seen today. Intermediate peers can easily modify or misroute traffic. Sybil attacks [42] are one of the most well-known threats to P2P systems. Named after the pseudonym used in a well-known study on multiple personality disorder, it involves creating multiple personas within the P2P network to be in a stronger position to perform malicious activities. By having a stronger presence in the network, attacks can misroute traffic, efficiently cutting off selected targets completely (also known as an eclipse [160]), amongst others. This has led to research of reputation-based prevention techniques, using either the overlay as storage (for votes, feedback and complaints) [37] [180] [85] [5], or by distributing the information as part of the network signaling [183] [140]. Although these techniques improve the security and reliability of the P2P networking, trust remains an issue.

Privacy in P2P systems is also often an issue. By contributing to the network, end-users run a higher risk of exposing, either accidentally or inadvertently, sensitive information [54]. Without an anonymization mechanism, the identities of the participants are also more exposed, as there is no third party through which data is exchanged. P2P networks lacking the proper encryption also allows intermediate peers to follow (eavesdrop on) or even disrupt the communication. Furthermore, as services and data can be provided by anyone, it can lead to irresponsible behavior on the part of the providers. Today we see viruses and other malware being distributed with the help of P2P file-sharing systems.

Operations over collections of content, such as searches, are also problematic. Per definition, each node in a P2P system has only a partial view of the network, and for a complete query, the request must be processed by multiple nodes. This is problematic in structured P2P networks in particular, where the precise routing algorithms are not well suited for the inexactness of common content queries. Although innovations such as range queries [12] [158] [58] and efficient multi-attribute searches [51] [156] [62] [21] [114] have been developed, P2P systems are still seen to excel best with rather simple bandwidth-intensive applications.

2.2 Public-key cryptography

Traditional computer cryptography is based on using a cryptography function *f()* that transforms a piece of data, *plaintext*, into encrypted data using a secret key S. The same key is used to transform the encrypted data back to its original format, hence often referred to as *symmetric* cryptography. The process seems intuitive as it is similar to real-world message concealment methods used since ancient times. Although symmetric cryptography does well in protecting the data from eavesdropping, it requires that the secret key is known by the parties exchanging data. It essentially requires that there is a pre-established relationship between each of the communicating actors, and that each one is trusted not to expose the secret to third parties as it would invalidate the whole system.

To solve this problem, a public key, or *asymmetric*, cryptography was proposed by Ralph Merkle in 1974, although first published by Whitfield Diffie and Martin Hellman in 1976 [40]. By leveraging the characteristics of exponentiation, these systems allow two previously unknown parties to establish a shared secret over an insecure channel. This key-negotiation scheme introduced the notion of using a key pair with a mathematical relation, of which one part is kept secret while the other can be exchanged over an open channel. The latter part is used for encrypting while the former, secret, part for decrypting.

The first practical public key cryptography scheme was published in 1978 by Rivest, Shamir and Adleman, today known as RSA [142]. The secret, *private*, and public key pair in RSA is chosen based on mathematical relationships to two randomly chosen prime numbers p and q. The public key consists of the two numbers e and n, where n is the product of the chosen prime numbers and e a co prime with $(p-1)(q-1)$. The private key d is calculated to satisfy an inverse relationship with these, as shown in Figure 2.5. The public key can now be openly distributed and used to encrypt information, which can only be encrypted using the private key. But the reverse is also true; the private key can be used to encrypt information that can be decrypted with the public key. Although the two keys do share a mathematical tie and can be deduced from each other, the security of RSA is based on the computational difficulty of factoring large primes. Choosing sufficiently large values for the two prime numbers makes it practically impossible to determine the private key from the public.

Choose two random large primes, p and q. Compute

$n = pq$

$\varphi(n) = (p-1)(q-1)$

Choose the public key e as

$1 < e < \varphi(n)$ and $gcd(e, \varphi(n)) = 1$

The private key d is now

$d = e^{-1} \bmod \varphi(n)$

Encrypting a message m to cipher c

$c = m^e \bmod n$

Decrypting a cipher c

$m = c^d \bmod n$

Figure 2.5. The relationship between the public and private keys in RSA, and how encryption and decryption is performed.

RSA solves the problem of a pre-shared secret as the public key used to encrypt information can now be openly distributed. But due to the inverse relationship of the two keys, it also offers another important feature; the ability of verifying the source of data. As content encrypted with the private key can be decrypted using the public key, messages can be digitally *signed* using RSA. The message (or a digest of it) is encrypted by the author using the private key and added to the message. The recipient uses the public key to decrypt this attachment, compare it to the message content and thus verify that the sender is the owner of the private key. The recipient can therefore not only verify the source of the data, but also its integrity.

Asymmetric encryption schemes offer many advantages over symmetric ones, but also some drawbacks. As the keys share a mathematical relationship, they can be produced from each other. As stated, the security is based on choosing large enough prime numbers which makes it practically impossible. The process is also computationally much more demanding, although in practice most implementations take a hybrid approach, where the content itself is encrypted using a symmetric key which is appended, encrypted using the public key.

2.3 Key-based identity schemes

As public key encryption offers the possibility of verifying the source of messages, these have been widely adapted for different identity schemes. Public key identity schemes are based on binding a real-life identity to a public key pair, assuming that only the rightful owner of the identity is in possession of the private key. As the key pair essentially becomes the identity, we can securely verify the source of any information published and authenticate peers without the involvement of a third-party trusted service.

As the public key (essentially a large number) is unintuitive for people to manage, most applications use a mechanism for mapping human-readable identifiers (names) to the actual keys. This can be done through third-party certification or by a leap-of-faith procedure on first contact. Third-party certification is achieved by having a trusted third party signing a piece of data (*certificate*) which provides the mapping of an identifier to a public key. The security is based on having the trusted third party's identity (public key) distributed beforehand (often pre-installed in applications). Leap-of-faith is the process of storing the public key locally on first contact.

2.3.1 Transport Layer Security and HTTPS

One of the most well-known public key identity schemes in use is the site certification used in the secure version of the Hypertext Transport Protocol (HTTP). HTTP Secure, HTTPS [141], is the technology for wrapping HTTP connections within Transport Layer Security (TLS [39]) connections. The TLS protocol supports both client and server authentication using electronic certificates. Modern web browsers come pre-packaged with a number of public keys of trusted certification authorities. These issue certificates for different web sites, which are presented to the user when visiting their sites. Certificates may also be *chained*, signed by multiple keys. The resulting system follows the X.509 public key infrastructure

(PKI) standard defined by the ITU-T.

Although HTTPS has been a key enabler for the modern web ecosystem and proven over time to be resilient to a number of attacks, it has received its share of critique. Aside of a few exceptions, the critique has, however, concentrated on the *use* of the protocol, rather than the protocol itself. Without clear implementation guidelines, implementations have often trivialized the authentication of the protocol, using it primarily as a solution for securing only the data transfer. Warnings of invalid or irresolvable server certificates have often been minimal, causing all but the most scrupulous users to ignore them [151]. This has enabled man-in-the-middle attacks and impersonation, allowing attackers to gain access to valuable credentials. Furthermore, as the protocol can be used in conjunction with normal HTTP, web service providers often utilize it for the most sensitive parts (such as login forms), while other data access (such as image loading) is done through unsecured connections. This creates opportunities for session theft for anyone that is able to listen in on the traffic (as is possible, for instance, in open wireless LAN networks).

Furthermore, the HTTPS model can be seen as outdated with regards to the economics of the modern web. We have long since passed the time when a single web site contained content from a single producer. Today, web sites may include content from a myriad of sources such as advertisements and user-generated media. The HTTPS model, however, only authenticates the hosting server, not the actual source of the content, making it impossible to independently judge the trustworthiness of it [104]. The lesson to be learned from HTTPS is thus that when creating a security solution, we should be careful not only when designing the protocol, but also pay attention to how it is implemented and used.

2.3.2 PGP

Another notable public key identity scheme is Pretty Good Privacy (PGP) best known for its use for securing e-mail [24]. Although current versions support certificate authorities, the first versions were based on utilizing a web-of-trust (WOT) for binding identities to keys. As a public key identity scheme, PGP is based on each user having a key pair of which the public part is made publicly available. However, without a trusted certificate authority anyone is able to publish keys which they claim belong to a specific person. The WOT scheme is based on other users verifying (or attesting to) a key belonging to the person it claims to.

When entering the system users ask others, already using PGP, to electronically sign an identity certificate containing the public part of their key and information about the person (name and validity). This signed certificate verifies that the key actually belongs to the real-life person with the given name. These peer signatures are commonly agreed upon in person, tying the scheme to actual human relationships. As the signing peers also have had their identities verified similarly, it creates a network between the users of PGP which can be traversed to find a link, *path*, between ourselves and a previously not encountered identity. This creates an identity verification scheme based on social connections and trust.

2.3.3 Opportunistic personas

The Opportunistic Personas [10] concept provides an interesting alternative to traditional identity management. Based on a model similar to PGP, where users use self-signed identities, it does not emphasize on establishing certain knowledge of the real-life owner (persona) of an identity. Instead, it is based on forming an opinion using our history with that actor. Traditional identity schemes try to tie an electronic identity to a real-world actor, either through third party certificates, WOTs or out of band communication. How we interact with that identity is based on our attitude towards the presumed owner. The Opportunistic Personas scheme differs as it tries to remove the dependency on both that, somewhat fragile, link between the presumed owner and the identity, as well as what we think we know or assume about the owner. Instead it proposes that applications use only knowledge from previous encounters to adjust their attitude towards an identity.

The full potential of the scheme is seen when it is used across applications to automatically record our history, or *track record*, with other actors. For instance, placing voice calls to a persona could automatically whitelist and prioritize emails received from the same actor, as we are likely to have a relationship with it. Although the scheme has obvious security issues and is not suited for all applications, it can provide a practical and more deployable model for others.

2.3.4 Identity revocation

A problem common for all public-key identity schemes surfaces when an identity is *compromised*, i.e., when the private key is exposed to an unauthorized party. As the secrecy of the private key is the cornerstone of the security in public key cryptography, a compromised identity becomes essentially worthless. Furthermore, in case the compromised identity is of a trusted authority, everything verified by it becomes questionable. Although there does not exist a method for *regaining* an identity after it has been compromised, modern system include mechanisms to minimize the risk and mitigate the damage. Electronic certificates issued by trusted authorities are given a specific life time, which results in compromised identities to be invalidated after they expire. This period (start and end dates) are embedded within the issued certificate to prevent tampering.

Most systems also have a revocation mechanism for explicitly blacklisting known compromised keys. However, these require each entity in the system to actively update its database, which can be problematic in distributed systems. Web-of-trust schemes can also be used in combination with out-of-band communication and real-world relationships to invalidate identities in decentralized systems. As these are based on information from only partly trusted sources (the opinions of others), these should be used carefully. To minimize the risk of an identity becoming compromised, systems can be built with using a layered approach. The real, or root, identity is stored in a highly trusted location (for instance a tamper-proof smart card), and used only to sign self-generated secondary identities. These secondary identities, often with short life times, are used for day-to-day activities, and can be handled more freely.

2.4 The Session Initiation Protocol

The Session Initiation Protocol (SIP) [146] is an Internet Engineering Task Force (IETF) standardized text-based protocol for establishing and controlling multimedia sessions. SIP is used to negotiate session parameters, such as media encoding and transport addresses, and to update these as needed. SIP has proven to be functionally diverse and has gained wide acceptance, with several open source and proprietary voice over IP (VoIP) and instant messaging (IM) clients available. SIP end-points, called User Agents (UAs), can exchange data directly, but in most cases a network of SIP servers are used. These help in locating other users, routing messages and setting up direct connections.

SIP users are referred to by an email-address- like identifier, the SIP Address of Record (SIP AOR). The SIP architecture defines a component called the Location service and Registrar for keeping track of the location of users. As the SIP identity is based on domain names, it is common for each domain that provides SIP services to host its own registrar server. As the location of users change over time, users *register* with this server (located by a DNS service record), announcing their current location. To establish sessions, the SIP clients use the Location services to reach the other user. The registrar servers are thus responsible for an important part of the system security, as they manage the identities and authenticate UAs, preventing identity spoofing and theft.

The SIP architecture also specifies another type of server, the SIP proxy server, which can be used as a gateway for users either for administrative reasons (network security policies might block certain types of traffic) or to ease the burden on the clients. These proxies commonly handle all signaling from and to the client, and are thus able to modify the session parameters. This can be used, for instance, to set up media channels for clients within closed networks, but allows also easy access to manipulate the behavior of SIP applications. By creating custom SIP proxies, it is easy to add new functionality (such as message filtering or statistics) or create gateways to other communication networks (such as to the public switched telephone network, PSTN). Figure 2.6 illustrates the architecture of SIP.

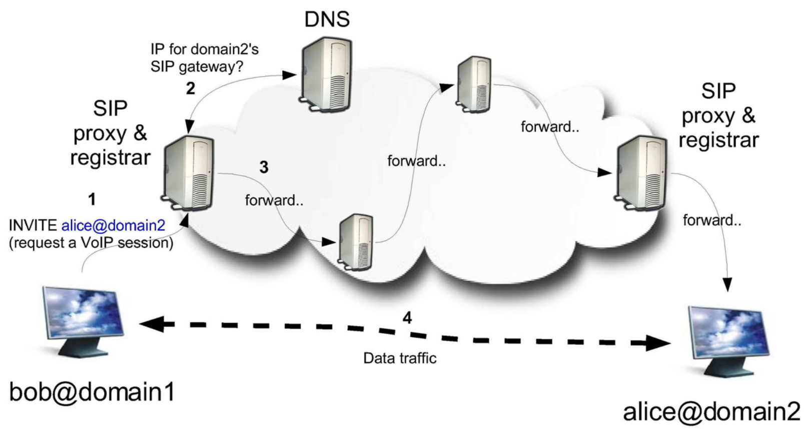


Figure 2.6. Illustration of the SIP architecture. Users uses proxies and registrars to forward signaling.

The standard SIP session begins with both parties of the session registering, using the REGISTER message, with the Location service of their domains. As SIP can be used both over reliable (TCP) and unreliable (UDP) transport protocols, the REGISTER message is acknowledged with a response containing an HTTP-like *response code*. This code indicates whether the registration was successful (code 200), or whether password authentication is required or an error occurred. Following a successful registration, the initiating user sends an INVITE message addressed to the other party through its proxy. The proxy will locate the SIP proxy of the recipient's domain and forward the message, possibly adding parameters, such as Via: entries used for routing responses. The message is further forwarded to the intended recipient, who answers with a response indicating whether the session (such as a VoIP call) is accepted or not. This is routed back to the initiator who, in turn, responds with a final ACK message acknowledging the response. The responding user (and proxies in between) may also return intermediate responses used to track the progress of the session establishment. These (such as the code 180, Ringing, response) can be used by the SIP UA to simulate the dial-tone behavior of PSTN.

```
INVITE sip:bob@p2psip.hiit.fi SIP/2.0
Via: SIP/2.0/UDP 127.0.0.1:6060;rport;branch=z9hG4bKdtbjuppq
Max-Forwards: 70
To: <sip:bob@p2psip.hiit.fi>
From: "Eve" <sip:eve@p2psip.hiit.fi>;tag=psjgk
Call-ID: odbydyiaylbjmdx@hafnium.pc.hiit.fi
CSeq: 907 INVITE
Contact: <sip:eve@127.0.0.1:6060>
Content-Type: application/sdp
Allow: INVITE,ACK,BYE,CANCEL,OPTIONS,PRACK,REFER,NOTIFY,SUBSCRIBE,INFO,MESSAGE
Subject: Meeting
Supported: replaces,norefersub,100rel
User-Agent: Twinkle/1.4.2
Content-Length: 303

v=0
o=twinkle 1934341310 1078326224 IN IP4 127.0.0.1
s=-
c=IN IP4 127.0.0.1
t=0 0
m=audio 8000 RTP/AVP 98 97 8 0 3 101
a=rtpmap:98 speex/16000
a=rtpmap:97 speex/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:0 PCMU/8000
a=rtpmap:3 GSM/8000
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
a=ptime:20
```

Figure 2.7. An example of a SIP INVITE message for initiating a VoIP session. The initiator and recipient are highlighted. Session media formats and transport addresses are contained in the body of the message.

The session parameters are conveyed within the INVITE message and its response. The initiating user populates the INVITE message with information on which transport-level address it is expecting the media traffic as well as the codecs or formats it supports. The responding user inspects these to find suitable ones that are supported by both, and sends these, along with its transport-level addresses, in the response. Figure 2.7

displays a sample INVITE message containing parameters for a voice call. Figure 2.8 shows the messages exchanged in a complete call set-up in SIP. In addition to basic session management, SIP has a long list of extensions to support features such as presence [143], instant messaging [29] and file transfers [28].

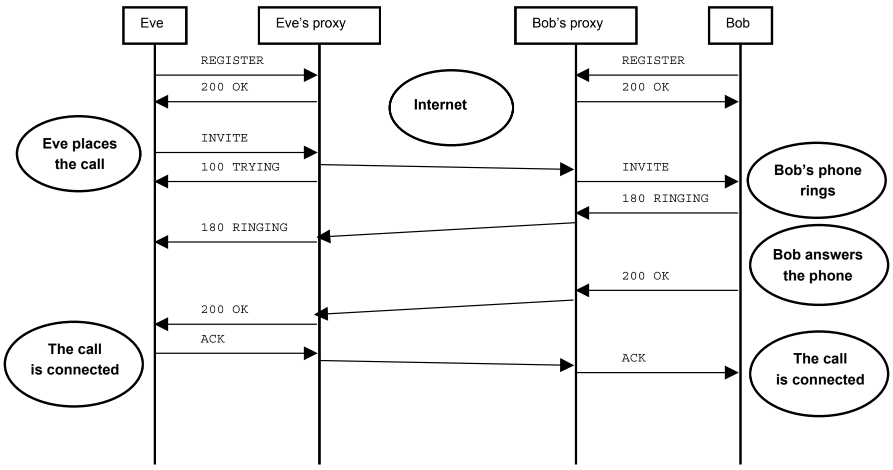


Figure 2.8. Sequence diagram of the messaging involved in establishing a SIP session.

2.4.1 SIP security

The SIP protocol was designed as a simple, easy to use, open protocol for multimedia sessions. The emphasis was on creating a protocol that is easy to understand, versatile and extendable, rather than a strict, secure, communication platform. The base specification of the SIP protocol contains a number of security vulnerabilities that can be exploited by different parties [53]. These touch on almost all areas of communication, including authentication, privacy and integrity. Although security extensions have since been proposed (and standardized), all systems do not implement these, or can be tricked into degrading the security to a lower level.

SIP uses for data transport protocols such as UDP and TCP. This means that it naturally inherits all the security threats associated with these, including service denial through TCP syn flooding. However, the base specification does not require the use of any additional transport security, leaving the signaling and media traffic susceptible to eavesdropping, interception and modification by common networking tools. Furthermore, common SIP deployments rely heavily on the DNS system, leaving them vulnerable also to DNS-based threats. This can be clearly seen in the SIP authentication procedure.

SIP authentication is usually based on a pre-shared password. Users connect to the registrar server of their domain and prove themselves using an HTTP digest authentication [50] exchange. This application-level authentication covers only specific fields of the signaling messages, and only certain messages. Moreover, it only authenticates the SIP client to the server, providing ample opportunities for different man-in-the-middle attacks. The registration, and thus identity, of the client can be captured using message replays, legitimate calls can be disconnected or redirected and unauthorized calls can be made. This can be used to launch DOS attacks against a user or monetary loss through billing attacks [181]. As the clients (as well as intermediate SIP proxies) rely on the DNS system, the attacker need not even physically be in the same network, but may utilize a DNS vulnerability such as cache poisoning to capture the traffic.

SIP security is, however, evolving. The use of both end-to-end as well as hop-to-hop security has been standardized, although not mandated. These rely on technologies such as TLS [39] and S/MIME [138] to protect the confidentiality and integrity of the signaling. Many SIP UA clients also support encrypted media formats such as Secure RTP (SRTP) [18], which prevents eavesdropping on the sessions.

2.4.2 SIP and VoIP SPAM prevention

As SIP has emerged as the prevailing open standard for Internet-based communication, many have feared that it will shortly be overwhelmed by the same problems as traditional email. SIP (and VoIP in general) SPAM prevention has subsequently been vividly discussed, resulting in a number of proposals as well as commercial products. The issue was noted also by the people responsible for the SIP base protocol, resulting in an Internet RFC discussing the issue [145]. It addresses the issue quite holistically, identifying different types of SPAM (Table 2.1) and listing a number of high-level conceptual solutions. The key observation in this analysis is that a strong identity scheme is needed to enable most approaches. Without identities that have value, spammers can easily generate new ones or impersonate others. Temporary, so-called *throw-away*, domains can also be registered at low cost to provide an abundant supply of SIP identities for spammers.

The importance of verifiable identities has been addressed in the IETF through various proposals. In [81] the authors describe a simple scheme where SIP proxies add information to the signaling indicating whether

Table 2.1. The possible types of SPAM in SIP systems [145].

Type	**Description**
Call	Unwanted VoIP calls. Includes marketing calls both by humans and pre-recorded messages, scams and identity theft through social engineering.
IM	Unwanted instant messaging. Marketing and scams, impersonation, link baits.
Presence	Requests for permission to follow ones presence may include marketing messages.

they have verified the sender (for instance through password authentication). However, this scheme only applies to specific trusted environments as these headers can easily be forged by malicious proxies. A more secure and scalable scheme for verifying the authenticity of messages is presented in [132]. This scheme builds on public-key cryptography, requiring domains to electronically sign the messages originating from their users. A hash value is calculated over certain immutable values in the messages by the domain's internal proxy. Before forwarding the message, the proxy signs this hash with a domain-specific key, and adds the signature as well as instructions on how to retrieve the domain's public key. The receiving domain is expected to fetch this domain key and check that the message has not been tampered with during transit. Furthermore, by signing the messages, the source domain proxy is assumed to have authenticated the user, which guarantees that a message was actually sent by the user it indicates. However, these schemes address only the verification of an identity within the context of that identity's domain, and can be circumvented by using rogue domains or domains that easily provide new identities.

A secure identity scheme is only the start of SPAM prevention in SIP and other communication systems. A number of techniques for dealing with SPAM have subsequently been proposed. These can be divided into one of three general categories: *i Non-intrusive* methods that operate either in the network or at the end-user's client. These work without interacting with either the initiator or the responder, halting SPAM before it reaches its target. *ii Caller interactive* methods that interact, or challenge, the caller in some way to prevent (or make it costly) to distribute SPAM. *iii Intrusive* methods that require the responder to interact with the system. Table 2.2 presents an overview of these.

Some of the proposed VoIP SPAM prevention techniques are quite gen-

Table 2.2. Schemes for SPAM prevention in Internet communication systems.

	Type	**Description**
Non-intrusive	Content filtering	Filter instant messages and files based on static analysis of the content.
	Black / Whitelists	Prevent or allow only specific users.
	Call patterns	Detect suspicious communication patterns in VoIP or IM indicating non-human or unwanted communication.
Caller interactive	Turing tests	Present different sorts of challenges to the caller to verify that it is a human.
	Resource dedication	Provide the caller with computational puzzles or other resource-demanding tasks before connecting the call.
	Query intent	Query the intent of the communication prior to connecting a call to repel (honest) telemarketeers.
Intrusive	Feedback	The responder provides feedback after the call on whether it was SPAM. Used in conjunction with black lists or reputation systems.
	Payment at risk	Before establishing a connection, the caller commits to pay in case the communication is SPAM. Requires feedback from the responder.

eral, adopted from email systems, such as black- and whitelisting. The crucial difference between VoIP and email SPAM prevention is naturally the real-time nature of VoIP. Static content analysis (which traditional email SPAM prevention largely relies on) cannot be used as such. But on the other hand, the real-timeliness of VoIP allows us to analyze the source of the call more thoroughly, even interact with it before passing the call through. Call signaling analysis could be seen as the nearest VoIP equivalent of email content analysis. For instance, the authors in [113] propose a system which uses different call characteristics to identify spammers based on assumptions such as that spammers usually do not receive calls, do not place more than one call per recipients and that the call-terminating partner is usually the same. The scheme presented in [178] builds on some of the same principles, but adds user feedback combined with machine learning algorithms to produce more accurate results.

Following the development of the email ecosystem, where SPAM is certainly not hand written for each recipient separately, VoIP SPAM is feared to become mostly based on playback of pre-recorded messages. However, as the VoIP calls are real-time, different types of Turing [171] tests (able to detect automated calls) present an interesting way to combat SPAM. These can either present the caller with a question, or *puzzle*, which is easily solved by a human, but hard to automate. These can be delivered within the call (e.g., a voice synthesizer asking to repeat a text or press certain dial buttons), or as an instant message or image file (e.g., a CAPTCHA [174]). Alternatively, the voice pattern can be analyzed to detect speaking patterns abnormal for normal humans. The authors of [137] present a system where the caller's reactions to a pre-recorded greeting message is analyzed to detect abnormal speaking patterns, such as double talk. These Turing tests can be made to initiate automatically before the user is notified of the call, transferring the call to the user only after they have been passed.

2.5 Peer-to-peer SIP

Although SIP is not bound to a server-based architecture, for any sizable system a network of trusted servers needs to be deployed. This creates administrative burdens, introduces single points of failure, as well as lucrative targets for attacks. Furthermore, the traditional model does not allow for ad-hoc sessions in isolated networks as we need to be able to connect to the different components of the SIP architecture. These, amongst others, are arguments that have ignited the development of fully decentralized, peer-to-peer (P2P), SIP systems. These are referred to as P2PSIP systems. There has been a number of proposals, mainly from the research community concerning experimental and proof-of-concept systems. In the following we review a number of these systems, as well as provide an overview of the standardization efforts of P2PSIP in the IETF.

2.5.1 Decentralized SIP

The SESSI (Seamless Service Interworking in Heterogeneous Mobile and Ad-Hoc Networks) project at Helsinki University of Technology developed as part of their research a prototype [159] [108] of a SIP proxy suitable for ad-hoc and P2P scenarios. The proxy used the service location protocol (SLP) [61] to establish sessions in ad-hoc networks, and the design allowed interworking with global networks [84] as well. The security was based on public key cryptography, with strong authentication and a relatively complex service-oriented permission model. The SESSI framework provided an early model for decentralizing SIP (and other services) in ad-hoc scenarios, but was not focused on creating one that could seamlessly move between these and global networks.

The SIPPeer adaptor [161] [83], developed at Columbia University, replaced the SIP Locator functionality with a Chord-based DHT which was established between the participating nodes. Furthermore, SIPPeer offered advanced features such as offline messaging, directory services and Network Address Translation (NAT) traversal. However, security had not been a key research goal and authentication used an email-based validation scheme, and the session security was not addressed. In related research at Columbia University, the use of an external DHT as SIP proxy replacement was examined [162], with some ideas integrated into SIPPeer. Specifically, support for the OpenDHT [127] public service, run on PlanetLab [134], was developed. The integrity of the data was protected using public key signatures.

SoSIMPLE is a SIP-based P2P communication model developed at the College of William and Mary [23]. SoSIMPLE is based on using a DHT as a distributed SIP proxy, similar to the work at Columbia University. It also touched on advanced topics such as NAT traversal and presence management, but did not provide a clear solution for security. Similar research was done at the Nokia Research Center [115], however, concentrating on the feasibility of such systems for mobile devices. Although the architecture was similar, focus was on experimenting with prototypes on mobile phones, examining the feasibility through call set-up time measurements.

The research within the DECICOM project at the University of Oulu has continued on the same track, contributing much to the current understanding of the feasibility of P2P communication systems in mobile environments. Although the research started with P2P SIP, using an implementation based on a (now expired) IETF protocol proposal, it has since evolved into other application domains, including P2P web services [106] [105]. The focus has not only been on the architecture, but also the overall feasibility of P2P applications in mobile environments in terms of usability, energy consumption and cost. This has resulted in extensive analysis on energy consumption as well as the effect different types of overlay algorithms and conditions have on mobile systems [88] [130] [128].

Part of the work in the DECICOM project was done in conjunction with Hautakorpi et. al. at Ericsson Research Nomadiclab. Based on the same protocol as the prototypes constructed in Oulu, the research was extended towards a broader view of distributed networking. The application support was extended from simply P2PSIP to a generic framework for applications such as HTTP, email and DNS [63]. Furthermore, integration of P2PSIP systems with commercial networks (IP Multimedia Subsystem, IMS) was analyzed [65].

The P2PNS system presented in [19] provides an interesting model for P2P computing. It describes a generic name-lookup service which uses a two-part process where the identity is first resolved to an overlay NodeId, which is further mapped to an actual IP address. It is also one of the first to explicitly plan for application other than SIP from the beginning, as well as presenting a security scheme for fully distributed networks without a centralized CA (based on cryptographic puzzles).

2.5.2 RELOAD

Recently the IETF has recognized the need for a standard for a P2P SIP protocol, and is currently working on standardizing the data protocol and architecture in the P2PSIP working group [74]. The current version of the P2PSIP protocol draft defines a highly modular framework supporting different applications (called *Usage*s) as well as overlay network types [79].

This framework, called REsource LOcation And Discovery (RELOAD), has an architecture which consists of the modules Message Transport, Storage, Topology plug-in, Forward and Link Management, depicted in Figure 2.9. This architecture provides a very general model for overlay networking, in which the implementer can choose between different types of routing algorithms (the Forwarding and Link Management module) without affecting the other parts of the system. This creates an interoperable and reusable framework that can be adopted for a wide range of distributed applications. As stated in [79], SIP is only one possible usage, and there have been proposals for usages such as conferencing [95] [69] and Simple Network Management Protocol (SMNP) [131].

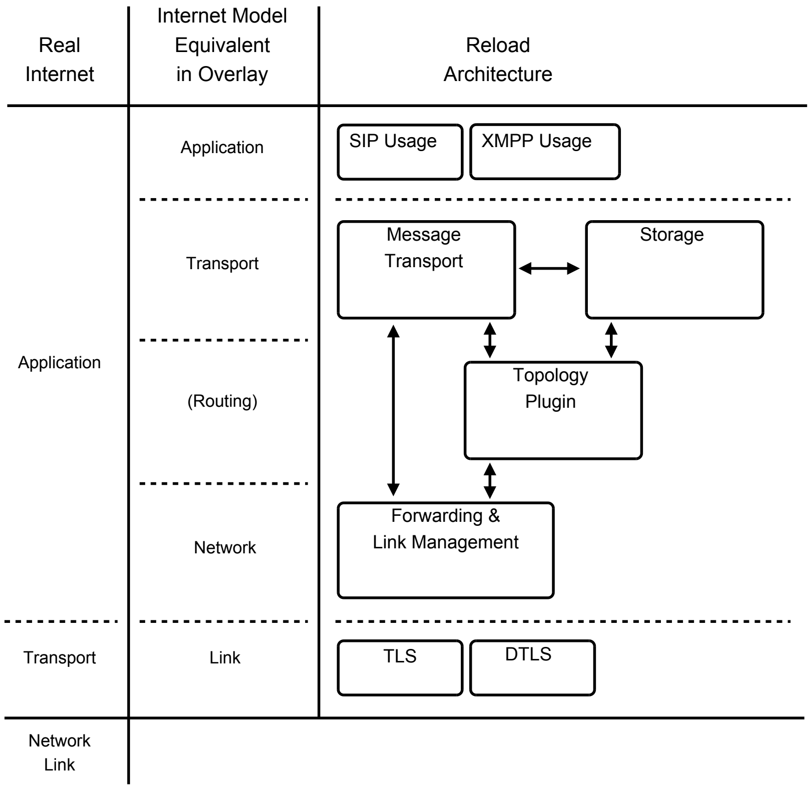


Figure 2.9. The RELOAD architecture, adapted from [79]. RELOAD creates a virtual network stack by separating the Topology plug-in and Link Management into their own modules.

Hosts in a RELOAD instance are assigned a unique NodeId, which is used for addressing within the network as well as by the Topology plug-ins to construct the overlay. The Topology plug-in uses the Forwarding and Link Management module to establish and maintain peer connections according to the overlay algorithm in use. The Message Transport module provides end-to-end communication for the applications, Usages, by querying the Topology plug-in for routing information and using the Link Management module to transmit packets. The Storage module is responsible for storing overlay data.

The applications, Usages, are provided a generic API for storing and retrieving overlay data, discovering services and establishing direct connections to remote users. This API is independent of the underlying network structure, making it possible to upgrade routing algorithms and transport protocols without affecting the application itself. The security of RELOAD is based on strong identities. Each node maintains a cryptographic identity, which is used to sign all messages. Furthermore, the data stored in the overlay is also signed, and secure protocols (such as TLS and DTLS) are used for transport. The identities can be issued (signed) by a trusted third party or, especially in closed networks, be self-signed.

SIP-based applications use this framework for three primary operations (the *SIP Usage*): registration, lookup and connection establishment [80]. Registering a SIP identity (SIP Address of Record, SIP AOR) with the overlay is done by creating a data packet containing the SIP AOR and a NodeId (the identifiers used for nodes in the overlay) or another SIP AOR, through which the user can be reached. This packet is stored in the network under a key made from the hash of the SIP AOR. The session signaling is exchanged directly between peers. To establish this connection, the calling party fetches the data packet and sends an *AppAttach* message to the NodeId of the target. This initiates an Interactive Connectivity Establishment (ICE) [144] procedure, during which *address candidates* are gathered, and sent to the target through the overlay network. These address candidates are transport-level addresses (Internet protocol, IP, address and a UDP or TCP port) through which the peer is reachable. As a host can have multiple addresses due to multihoming and network address translation, of which not all can be used by a remote peer (in case of closed local networks or firewalls), the different address candidates are tried until a working connection is made.

2.6 The Host Identity Protocol

The Host Identity Protocol (HIP) provides a communication architecture that separates the network location from the identity. End-points are bound to cryptographically generated identities, which are dynamically mapped to network locations, providing transparent node mobility and multihoming. The HIP architecture is described in the IETF Request For Comments (RFC) document 4423 [121], with the protocol details and extensions currently being standardized by the HIP working group [72].

The identity in HIP, the Host Identifier (HI), is the public part of a public key pair, making the binding secure and self-authenticating. However, applications more commonly use a 100-bit hash of the HI, the Host Identity Tag (HIT), as an IPv6 address with an ORCHID class prefix [123] for referring to the end-point. This way applications using HIP do not address hosts as legacy applications do when networking. Instead, the communication is between the actual identity of the remote party, not the host on which it is assumed to reside. This creates a more secure and identity-focused networking environment. Furthermore, although most of the current implementations do operate on a host-level, the HIP protocol itself allows almost any network resource to be assigned identities. We could assign users, hosts, applications, services or even individual network sessions their own identity, which are all accessed in the same manner using HIP. This identity-based networking paradigm is highly relevant as the penetration of Internet connectivity increases, and becoming embedded in different types of devices (as in the Internet of things) [59].

HIP is architecturally located between the transport and network layer in the network stack, creating a new thin waist as illustrated Figure 2.10. The HIP stack translates application-used network addresses, the HITs or their IPv4 representations (Local Scope Identifiers, LSIs), into routable IP addresses with which it initiates a four-way handshake where the identities are verified and connection details negotiated. Data transfer with the end-point is done using any transport protocol (e.g., TCP or UDP), which is encapsulated for transit using a security protocol such as IPSec [82]. This can further be encapsulated for NAT or proxy traversal, which can be further assisted by third party relays (HIP Relays) and Rendezvous Servers (RVS). The NAT traversal of HIP is based on the use of UDP encapsulation, and the use of the Interactive Connectivity Establishment (ICE) methodology [144].

The translation from HI to network location is done using different types of lookup services. Although the traditional lookup service of the Internet, the DNS, can be used, due to the self-certifying properties of the identities used in HIP, more scalable and dynamic systems are possible. The lookup service can therefore even be based on different types of

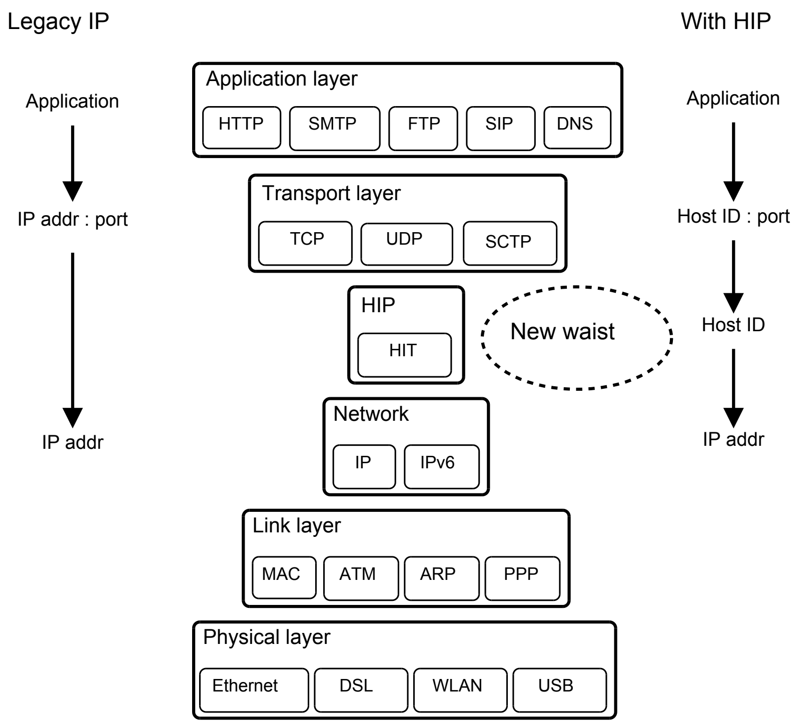


Figure 2.10. HIP creates a new thin waist between the transport and network layer of the networking stack. End-points are addressed using cryptographic identities instead of network locations.

distributed topologies consisting of ordinary (not controlled by a trusted third party) peers. Tampering is prevented by cryptographically signing the locator mapping with the respective Host Identity. There has been on-going interest in the scalability of HIP, with efforts put into distributing not only the lookup of identity-locator mappings, but also connection establishment. The original HIP architecture introduced the RVS and HIP Relay for enabling hosts between firewalls or strict NATs to connect. Although these can be distributed and are designed to maintain only a minimal amount of state, it still creates dependency points within the network that could potentially become overloaded. The Hi3 architecture [122] addressed the issue by introducing a fully distributed lookup and connectivity overlay based on Secure-i3 [9] [166]. In this architecture, hosts published the initial connection establishment packets to the overlay, where they are fetched by interested parties using a publish-subscribe model.

HIP is a maturing protocol with currently a number of implementations available for Microsoft Windows, Mac OSX, Linux and FreeBSD [170] [68] [67]. Also, Helsinki Institute for Information Technology (HIIT) maintains RVS servers for public use. HIP for Linux [68] is an open source implementation of the HIP protocol for Linux-based systems. The project has been developed at HIIT and features a number of extensions such as NAT traversal, mobility and RVS support. As HIPL is also available for Linux-based mobile systems, it was chosen as the platform for our prototype.

2.6.1 HIP overlay networking

As HIP provides a model where the identity of connection end-points are not tied to the network structure, the model itself can be adopted to different types of underlying network architectures. Recently the IETF HIP working group [72] has explored the possibility of combining HIP with overlay networks, resulting in the HIP Based Overlay Networking (HIP BONE) proposal [27]. The proposal defines a model that does not rely on the traditional Internet protocol addresses for locators. Instead, it utilizes different types of overlay networks for routing packets based on the HITs. This proposal is designed to enable overlay-based applications (such as P2P applications) to be better able to use HIP. These applications would often benefit from the security and mobility that HIP could provide, but due to their distributed nature cannot rely on the IP-based routing that HIP has previously required. The HIP BONE proposal addresses this issue by defining the required extensions for HIP to support overlay network-based routing.

However, the HIP BONE framework provides only a high-level model for HIP overlay networking, with the integration details left for application designers. One of the overlay networks which has been proposed as an instance for the HIP BONE framework is the IETF P2PSIP protocol, RELOAD. The RELOAD protocol (discussed in Chapter 2.5.2) provides an overlay network framework for different types of distributed applications, such as multimedia communication. The authors of [93] describe an alteration to the base RELOAD framework, which updates some of the functionality of the Forwarding and Link Management module to be based on HIP (and associated network security protocols). This draft also specifies a model for combining the identities used in HIP and RELOAD, as well as how the different RELOAD message types should be used to convey HIP-related parameters. However, it relies on relatively new extensions to HIP, most notably the multi-hop routing extension [25] and the data packet extension [26], and should be considered highly experimental.

2.7 Teredo

Teredo [71] is an IPv6 over IPv4 tunneling protocol designed to traverse NATs and other network middleboxes through the use of UDP encapsulation and *hole punching* techniques [49]. Teredo uses a public server, the *Teredo server*, for initializing the connection between the *Teredo clients*, removing the need for each client to have a public IP address. The Teredo protocol is designed to be simple and light, both in packet overhead and load on the Teredo server. The server load is kept minimal by only forwarding the initial packet and encoding the state into the Teredo IPv6 addresses. This has contributed to its popularity, and Teredo is currently available for a wide range of operating systems (including a built-in support in Microsoft Windows), with a number of organizations maintaining open, free, Teredo servers.

Teredo was developed as an improvement to the 6to4 [30] IPv4 to IPv6 transition protocol which required the hosts to obtain public, globally accessible, IPv4 addresses. On initialization, the client hosts, *Teredo clients*, contact a public Teredo server through IPv4 UDP, which assigns them a unique IPv6 Teredo address. Before assigning this address, the server will check the type of NAT (if any) the client resides behind (using a secondary source IP address for probing). The client address consists of five fields, as depicted in Figure 2.11. The first field is the Teredo prefix, `2001:0000/32` and the second the Teredo server's public IPv4 address. The third field contains a 16-bit flag field which is used to indicate the type of NAT the client is behind. The rest of the address contains the public IPv4 address of the client or its NAT and the port number used (as seen by the server). When contacting another Teredo client, that client's server address is decoded from the Teredo IPv6 client address, and an initial session set-up packet (*Teredo bubble*) is sent. This is forwarded by the server to the client, which decodes the initiating client's IPv4 address from the received packet, and tries to establish a connection by sending ICMPv6 ping packets directly to it (hopefully opening a route through any possible middleboxes in between).

Teredo does have a few drawbacks. The additional encapsulation and server relaying obviously adds a small amount of overhead compared to native IPv6 connectivity. More importantly, the NAT traversal method used is simple, and does not penetrate all types of NATs. Although extensions have been developed to the core Teredo specification to improve the

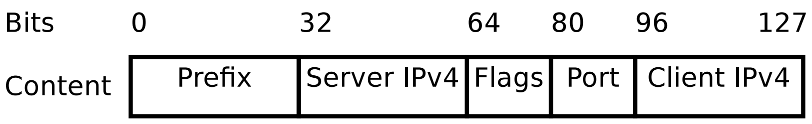


Figure 2.11. The Teredo client address contains the Teredo prefix, the server's IPv4 address, flags indicating NAT type and the client's public IPv4 and port. Adapted from [71].

NAT traversal capabilities, some undocumented while others, such as the SymTeredo proposal [70], not, they have not been officially integrated to the Teredo protocol.

2.8 P2P HTTP

P2P web services have received a fair amount of interest through a number of proposals [16] [52] [152] [44]. Web services is a term used to describe a service model where remote procedure calls are made over HTTP using technologies such as Simple Object Access Protocol (SOAP) for data exchange, Universal Description Discovery and Integration (UDDI) and Web Service Definition Language (WSDL) for service discovery. This model is based on a centralized lookup mechanism, where services are registered and found using a single service broker.

The proposals for P2P web services depict architectures where end-users, the peers, could act as both consumers and providers of services, using a distributed lookup mechanism for rendezvous. Although in line with our goals, these proposals concentrate on distributing the lookup, ignoring deployment details such as network obstacles (middleboxes) and certain security aspects (strong identities). Although suitable for limited environments, these do not scale to global networks.

2.8.1 Web caching

Distributed web caching and content sharing is a topic closely related to P2P HTTP. These systems focus on locating static resources (such as images or markup pages) using the HTTP protocol, instead of establishing a connection with an end-point for dynamic interaction. Although there are a large number of proposals in this area [157] [165] [110] [179], the Squirrel web cache [76] serves as a good example for demonstrating the key ideas as it provides a clear model that has been thoroughly analyzed [33].

The Squirrel web cache is based on creating a structured (Pastry-based [147]) overlay between the peers in which responsibility for content is divided equally between the peers according to the address, the Uniform Resource Location (URL), of the content. Requests are delegated to the peer responsible for that content, which can act according to two different schemes proposed by the authors of Squirrel [76]. In the *home-store* scheme the responsible node either returns the content (if cached) or fetches it from the source server. In the *directory* scheme the responsible node maintains a list of peers that have recently accessed the content, and forwards the request to one of those. In content distribution schemes, similar models can be used to distribute the load between the peers of the network. These schemes however, focus on the efficient distribution of the load, without considering connectivity issues or security, and are thus suited only for networks with trusted peers.

2.8.2 Personal web services

Nokia's Research Center has developed a personal mobile web server for Symbian Series60-based smartphones [119]. This Apache HTTPD [13] based software can serve both static content and dynamic, context-dependent, pages written in PHP[5] or Python[6]. These mobile web pages are thus able to serve as a personal information center integrated with the core mobile applications. For instance, information from the phone's calendar can be utilized as well as any photographs or short messages found on the device. Compared to the P2P web service proposals, connectivity has been carefully addressed, as it presents one of the greatest challenges in cellular environments. To reach the mobile web server, the system relies on a public server for relaying all data traffic. Although this adds to the latency, it is considered satisfactory for the intended use.

Opera Unite[7] offers a similar P2P web experience for desktop computers. Instead of a dedicated server application, it is based on JavaScript applications that are run within the Opera web browser itself. As with the mobile web server, it uses a centralized model for authentication and rendezvous, and does not provide an interface for external applications. It does embrace the concept of P2P web services by packaging the applications into so-called Unite service packages, and providing a catalog

[5]PHP: Hypertext Preprocessor http://www.php.net.php
[6]Python programming language, http://www.python.org
[7]Opera Unite, http://unite.opera.com/

service for easy distribution.

2.8.3 P2P social networking

With the overwhelming popularity of social networking services seen today, a number of projects have sprung up that aim to create decentralized versions of these services. Although these can be seen as instances of the personal web services discussed in Chapter 2.8.2, they are specially tailored for social networking only. These create a *social overlay* for different types of activities, instead of a generic HTTP network. The driving force behind these is often also rooted in the fear of the power and control the popular centralized social networking sites have over people, not so much in developing a more scalable solution for these services. As users become more dependent on social networking services, providing them with status updates, images and other personal information, concern has been raised over how this growing amount of sensitive data is being used. For instance, selling it to advertisers (to provide targeted advertisements based on interests) can provide a lucrative source of income. Furthermore, sudden changes in site policies could publicly reveal more data than users expect, which could lead to embarrassment or be used for social engineering and even identity theft.

P2P social networking projects such as Diaspora, GNUSocial and AppleSeed are creating rich web-based environments where users can connect and interact without handing information over to a third party. The data is stored locally, and shared only with trusted friends. This gives users more control over their on-line identity, but with a number of drawbacks. Users must host the social networking software on their local computers, and be able to connect with their friends to share. This ties the service to specific computers, and requires them to be regularly connected to the Internet. In practice, the users need to set up a personal web server running fairly complex software. Furthermore, these need to be publicly exposed to the Internet, allowing incoming connections, which can be problematic for normal home-users.

2.9 Bloom filters

Bloom filters, conceived by Burton H. Bloom [22], are space-efficient probabilistic data structures used to determine whether an element is part of a set. Although several variations exist, the basic Bloom filters use k different hash functions to compress the element's into a k- bit *signature*, which is stored in the m- bit Bloom filter.

An empty Bloom filter consists of m bits, all set to zero. When adding an element, we use the k hash functions to obtain k bit positions within m, which are set to one. As compressing a hash value into a single bit entry (essentially performing a modulo m) will cause collisions between the signatures of different elements, the Bloom filters are probabilistic in nature. We can therefore determine with certainty whether an element is **not** part of a set, but only within a certain probability whether it is part of it. To keep these *false-positives* low, the values of m (filter size) and k (number of hash functions) needs to be chosen according to the expected size of the set [177].

When adding a value to a Bloom filter, the probability that a single bit within an m- sized filter will be set by single hash function is $\frac{1}{m}$. After this single insertion, the probability that a randomly chosen bit is unset (not set by that hash function) is therefore $1-\frac{1}{m}$. When using k hash functions, the probability that a single bit remains unset after the insertion of a single element becomes then $(1-\frac{1}{m})^k$, and after n elements $(1-\frac{1}{m})^{kn}$ with the implication that the probability that the bit is set $1-(1-\frac{1}{m})^{kn}$. For an arbitrary value to be falsely seen as present in the Bloom filter, we need all k hash functions to produce a value that is set in the filter. Therefore, the false positive probability becomes:

$$p = (1-(1-\frac{1}{m})^{kn})^k \tag{2.1}$$

The advance knowledge of the number of elements any filter will store becomes especially problematic when these Bloom filters are passed and edited by multiple users. A too optimistic choice of values for m and k (small filter size) will easily result in overfull filters, rendering the content useless. On the other hand, a filter that is too large, in proportion to the number of elements stored in it, looses its efficiency and becomes essentially a bloated hash table. Being able to scale these filters dynamically has therefore, quite naturally, gained much attention with a number of more or less efficient schemes proposed.

Scalable Bloom Filters (SBF) [11] presents a scheme where Bloom filters are grown dynamically to keep the false positive probability within a specific bound. The filters are expanded adding new *slices* when the fill ratio (percentage of bits set) of the filter exceeds a pre-defined value.

These new slices are subsequently used for adding new elements, and determining memberships is done by checking all slices. In order to keep the false positive probability from linearly increasing, the fill restrictions are tightened for each successive slice (by increasing the size of the filters or lowering the fill ratio threshold). For instance, by doubling the size of each new slice, the total error probability will be the sum of a geometric series, for which we can choose the parameters so it converges at a specific value. Dynamic Bloom Filters (DBF) [57] presents a similar scheme, but with looser control over the growth of the total error ratio.

Although both schemes adds a degree of dynamicity, providing a more efficient way of using Bloom filters without knowing beforehand the element count, they do change the basic structure of Bloom filters. As the filters become structurally matrices, certain operations (such as unions and intersections) are complicated or impossible.

3. A secure peer-to-peer framework

Our research begun with the design of a P2P framework, whose primary purpose was to serve as a platform for studying applications in providerless environments. However, the architecture of the framework itself was also in focus, as we aimed to address the topic as broadly as possible, including deployability and cost, which relates to how the network, and therefore our framework, is built. The framework should be able to operate in different types of networking environments, with different types of data. As discussed in Chapter 1, the common approach to P2P networking is to create unique systems, each tailored for a specific application or environment. Our focus was not on a specific application, but rather an easily deployable utility that could be adapted to different uses.

As the trend in personal computing is clearly moving towards mobile environments and embedded devices, with location-independent connectivity, one of the key influences on the design was the requirements of these mobile, resource-limited, devices. Although the features, capabilities and processing power of mobile devices are growing exponentially, there are certain factors (such as cost of cellular networking and battery life) that will continue to limit what is feasible. This lead us to the following requirements that the design should fulfill.

Mobility and **strong identities**. The popularity of different types of Internet- enabled devices has shown that users are not tied to a single one, but choose one based on convenience, location or mood, to access the same service. As many of these, especially hand-held mobile devices, are often carried on person and tend to change network location frequently, the framework should support both identity, and network, mobility. We can not rely on devices being accessible at fixed locations for a long period of time, or that the network around them remains static. Therefore, we should not rely on trusted authorities to authenticate and secure the

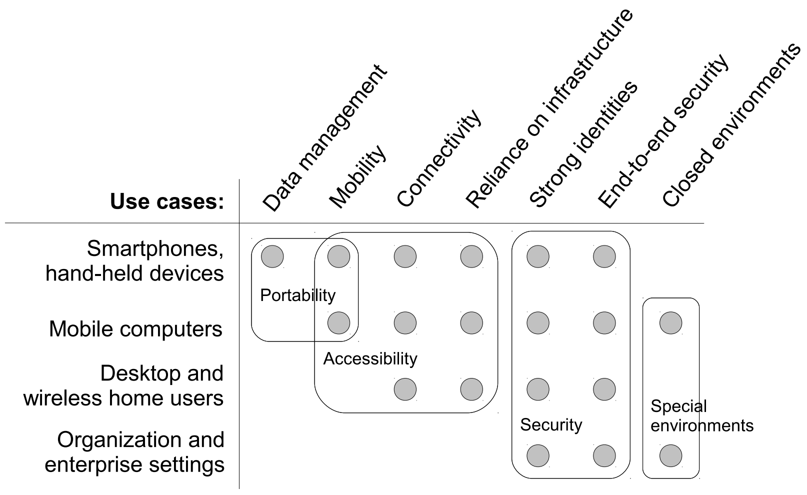


Figure 3.1. The design principles and their importance for different use-cases and environments.

users, but instead use a strong, self-verifying, identities.

Reliance on infrastructure and **closed environments**.. Aside from authentication, the framework should not rely on other infrastructure either, but be able to operate in completely closed environments. Connection establishment and peer lookup is commonly performed using public servers, or well-known peers. However, future networking environments may become more fragmented than today, for instance due to security concerns, or personal- and proximity networking. The framework can still use public nodes to ease connection establishment, but should not be completely reliant on them.

Connectivity and **end-to-end security**. Users should be able to connect, independent of their location. As a large proportion of the end-hosts may reside behind various network obstacles (such as firewalls or NATs), the framework should provide a solution for traversing these. We also need to provide multiple layers of security. Data may traverse through intermediate nodes, and should be protected against eavesdropping and in-transit modification. Finally, **data management**. The framework should be optimized for mobile devices, minimizing the amount of data these need to process, in order to spare memory and power consumption. Figure 3.1 summarizes these aspects, together with their importance for different types of users.

In the following chapter, we review the design and implementation of our framework in order to provide the context of the research. Lastly, we highlighting the technological advances of the framework compared to the state of the art.

3.1 System overview

By definition, a fully distributed P2P system treats all peers equal and relies only on these, the end-user devices and applications, to operate. However, to achieve the required level of security and connectivity in all environments, two additional roles were introduced in our design: the identity authorities and the overlay infrastructures. Following is a high-level review of the actors of the framework, illustrated in Figure 3.2.

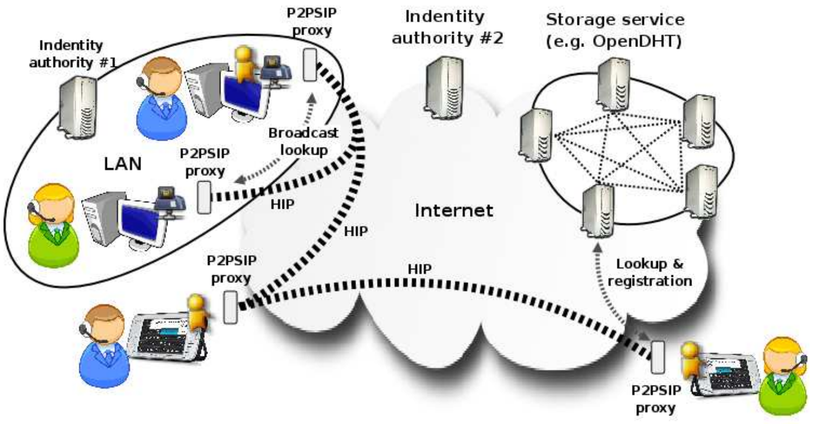


Figure 3.2. The components of the P2P framework. In addition to the users and peers, identity authorities and overlay infrastructures can be present. P2P connections are made using a data protocol such as HIP.

The **peers** are the ordinary end-users' nodes. These devices run either applications that are able to directly communicate with the overlay, or a P2P framework daemon that connects legacy applications to the system. Connections between peers are established using a suitable *data protocol*, such as HIP BONE (as explained in Chapter 2.6.1). Furthermore, the peers contain everything needed to manage their security themselves, which make them independent, and able to move between different types of environments easily.

The **identity authorities** are trusted nodes that solve the introduction problem in larger environments. These act as authorities, providing verification that an identity (a public key) belongs to a specific user through a digital certificate. Without these authorities, it is hard to keep track of identities, and impossible to verify new ones, leading to impersonation and misbehavior, as malicious activity is hard to attribute to a specific peer. Essentially these authorities ensure uniqueness and consistency of identities.

The **Overlay infrastructures** are collections of network resources that can be utilized for more efficient networking. These can be anything from simple on-line storage sites to public peers and overlay networks. These entities need to offer only semi-persistent storage in order to be useful.

3.1.1 Identity management

Our framework uses a strong, public key-based identity scheme, on which much of the security is based. Each user generates (or is provided) a public key pair which is used as their unique electronic identity in the system. This key pair is used for authentication during data session, and to electronically sign and encrypt the data packets stored in remote storage and overlays. As the integrity and confidentiality of data is thus ensured, we are less dependent on the quality of the underlying storage service.

However, as public keys are both inconvenient and unintuitive to manage directly, the system binds these identities to human-readable identifiers, *names*. Similar to how domain names are mapped to IP addresses, only these names are exposed to the end-user and the client applications when referring to other users. These identifiers follow an email-like *username-domain* structure, which allows for flexibility, as the uniqueness of a name can easily be controlled within the context of a specific domain. Furthermore, similar schemes are widely used in current applications, making it easier to adapt legacy applications to our framework. For securely mapping the identifiers to public keys, we use electronic certificates.

The identity authorities are trusted third-party certificate authorities used to securely introduce peers to each other. These authorities issue certificates presented during peer connections, that bind the public key of a user to the user's unique name. Compared to public-key infrastructures, identity authorities do not follow a hierarchy, and can be created at will, as long as both parties of a session trust the same. For instance, enterprises can maintain their own authority for issuing certificates to corporate-specific identifiers. At the same time, users can also trust the certificates issued by an authority created within a small group of close friends.

Although the scheme requires the users to trust the authorities behind the root certificates, it is not designed to provide trust between the users.

The goal is merely to provide a guarantee of the uniqueness of an identity in order to prevent identity theft and spoofing (much like the authentication scheme specified by OpenID [3]).

Furthermore, the identity authorities are not the only way in which peers can be introduced. The framework does not require that the identity certificates are issued by any specific authority, but support also certificates issued by other peers the user trusts, similar to the web-of-trust used by PGP. This flexibility enables identity mobility by allowing users to issue temporary certificates even of their own identity. For instance, instead of transferring ones identity to a rental device, we can just issue a certificate that *borrows* our identity for as long as we use the device. Alternatively, we can utilize third party identity- and key management systems. For instance, SafeSlinger[46] provides an easy-to-use and secure system for exchanging public keys within a group of users. These keys, or identities, can be accessed from other applications using their API.

Establishing actual trust in users is a complex and highly subjective decision, resolved on a completely different layer. The identity scheme only verifies that the name a user is using actually corresponds to his cryptographic identity, when processed within the framework. Figure 3.3 illustrates this process, and how identities are handled in our model.

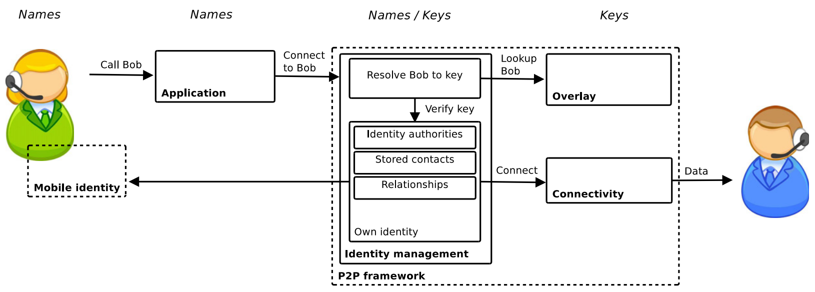


Figure 3.3. A high-level view of the identity handling. Users and applications use identifiers, *names*, when addressing remote peers. The identity management of the framework translates these into identity keys using the overlays. The identity of the local user is used to verify the mapping, using identity authorities, stored contacts or relationship information.

3.1.2 Overlay infrastructure

The overlay infrastructures are resources that provide lookup- and rendezvous services that assist peers in finding, and connecting, to each other, similar to what overlays and BitTorrent trackers do. The framework is not tied to a single overlay or type of network resource, and can even operate without any. The type of resources that may be used include both ones created specifically for the framework, such as overlay networks, and dedicated servers or public, well-known, peers. The requirement for acting as one consists of providing some storage space and offering a simple key-based interface for accessing data.

The framework defines a very basic set of operations that these resources must support. At a minimum, these include only the three storage operations *put*, *get* and *remove*. In addition to these, the framework can make use of signed operations, publish-subscribe services and message routing. The signed operations are designed to ease the burden on the clients when fetching data published by a specific peer. Instead of the client having to filter the response entries, the overlay can be instructed to do so before returning the results.

For best performance, the storage services should naively support an indexing scheme based on cryptographically generated keys, such as hash digests. These are primarily used for locating peers by retrieving self-published information packages that are indexed, for example, using the digest of the identity identifier. As the storage services are open and even unreliable (with regards to availability and eavesdropping), there are a number of security issues to consider. The data packages are signed, eliminating spoofing, but polluting the storage with invalid entries is still possible. The signed storage operations are designed to mitigate this threat, as the storage would only accept properly signed content and is able to filter the request results according to the signer. In Chapter 5 we discuss our privacy extensions which further address these issue by obfuscating the lookup keys.

3.1.3 Connectivity

As different environments and applications may require different types of data protocols and connection types, we do not limit ourselves to a specific data transfer protocol, but instead create a flexible model where we allow different types of existing (and possible future) protocols to be used. This modularity is however hidden from the client applications, presented only through a common interface. In our model, as we do not rely on overlay routing being possible, we need efficient methods for traversing the network obstacles present today that prevent direct connections.

Our focus has been on using the existing, stable, solutions Teredo [71] and the Host Identity Protocol (HIP) [121]. Currently these protocols are the most promising candidates for enabling P2P connections for ordinary end-users. Although competing technologies, such as routable IPv6 addresses and Mobile IP, has had some success, they are still inaccessible to many and do not solve network middlebox traversal completely.

By using HIP, we also relieve the application layer from solving a number of other issues. HIP is designed with security in mind, and provides strong authentication between end-points and encryption of the data traffic. Furthermore, HIP provides features such as connection mobility and multihoming which enables hosts to move between networks seamlessly. More recently, as discussed in Chapter 2.6.1, there has been work on standardizing extensions for HIP to support certificate-based authentication and construction of overlays based on the Host Identities. These show great potential for further improving the usefulness of HIP in P2P scenarios.

Teredo provides a widely deployed and slightly lighter platform for NAT and other middlebox traversal. Although lacking the security of HIP, combined with a transport security, it is a potent alternative for P2P connections.

3.2 Prototype implementation

A working prototype implementation was needed to gain sufficient experience and insight into potential issues of the framework design. We needed to asses the usability of applications adapted to our P2P model, and the feasibility and performance of overlays constructed using only simple, and possible unreliable, components (as described in Chapter 3.1.2). Furthermore, we needed a prototype which would operate in the same types of environments and devices we see future applications being used on, namely mobile networks and hand-held devices. The applications should portray the activities we expect users to engage in, not only demonstrate the technical performance of the framework.

This led us to design the prototype as a network proxy for existing, *legacy*, application, replacing the centralized service provisioning these rely on today with our distributed model. Although integrating the framework as a part of specific applications, or even constructing completely new ones could provide a more seamless experience, our approach made integration easier, allowing us to experience with more, and different types or applications.

The prototype was implemented on the GNU/Linux operating systems. Linux was chosen for its versatility, as it allowed us to experiment with applications both on personal computers as well as hand-held mobile devices and smartphones. The prototype was designed with the limitations of mobile Linux operating systems (such as Maemo [1] and MeeGo [2]) in mind, as an efficient background daemon. The prototype was implemented in the C language, with only a few external library dependencies, making it easy to port to different types of Linux environments.

The daemon acts as a network proxy, intercepting traffic from local applications which it maps, in an application specific manner, to our P2P model. In the following we will review some of the details of the implementation that are important for understanding how the experiments were conducted.

3.2.1 Architecture

The prototype was designed to be highly modular, making it easy to experiment with new components and features. This modularity was achieved by separating functionality into units offering clear interfaces for performing tasks, receiving updates and modifying the behavior of the module. For instance, following a successful peer connection establishment, an event is issued which modules can dynamically choose to receive. These can be used, for instance, to initiate data synchronization between the two.

At the core of the prototype is an asynchronous, multi-threaded, event engine (the **processor** module). The event-based architecture enables complex concurrency, while supporting also long-lived tasks through worker threads. Application events (e.g., signaling for a VoIP call) may result in a large number of networking operations, such as lookup and connection establishment, which would easily stall the whole system otherwise. Figure 3.4 depicts the architecture of the prototype's components.

The **ident** module contains the identity-related logic. It manages the locally stored identities and performs all identity-related cryptographic operations. It also provides an interface for utilizing identities from external identity modules, either as the primary identity, or as temporary ones by acquiring certificates authorizing it for a fixed period of time. This design allows for mobility, as we can use identities stored on smart cards

[1]The Maemo development platform. http://maemo.org
[2]http://meego.com

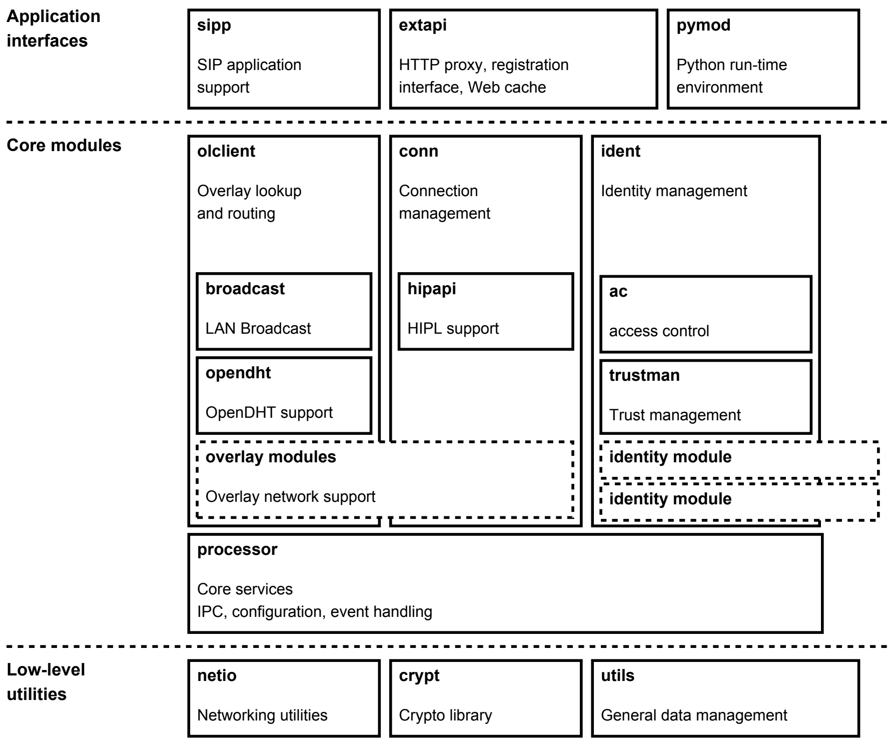


Figure 3.4. A high-level overview of the modules of the prototype.

or mobile phones while accessing the network from a untrusted computer.

In addition to managing the identity, the ident module keeps track of relationships, and provides access control and trust management based on these. Though sub-modules, it maintains a database of application-provided reports about encounters with external identities (users). These reports contain information such as the *context* of the encounter (e.g., voice call or email) and the *trust* gained or lost, which is summarized and made available for all other applications using the system. This allows applications to better identify unknown peers, as they are provided with a track record- *where* or *how* that peer has been encountered before, and what the peer's behavior has been.

The **conn** module manages the data exchange with remote peers. It provides a simple interface for either packet- or stream-based communication using only the peer name and a *service* identifier. These service identifiers refer to specific application instances, much like port numbers in TCP. And in a similar manner, modules need to register handlers for them in order to receive data. Communication with remote peers is thus completely hidden behind a single interface, independent of the type of connection used. The prototype supports currently connections made using HIP, plain IP and Teredo.

The **olclient** module provides an abstraction for interfacing with the overlay infrastructures. The actual interaction logic of any overlay is packaged into sub-modules, which provide a common interface for the overlay operations discussed in Chapter 3.1.2. The olclient module issues each request on each one of these sub-modules in parallel, returning the results (e.g., for fetching resources) as they become available. Currently we have implemented sub-modules for OpenDHT, for a proprietary web-based hash table lookup, and for local area network (LAN) broadcast-based lookup. These provide lookup for both global deployments as well as closed, ad-hoc, networks.

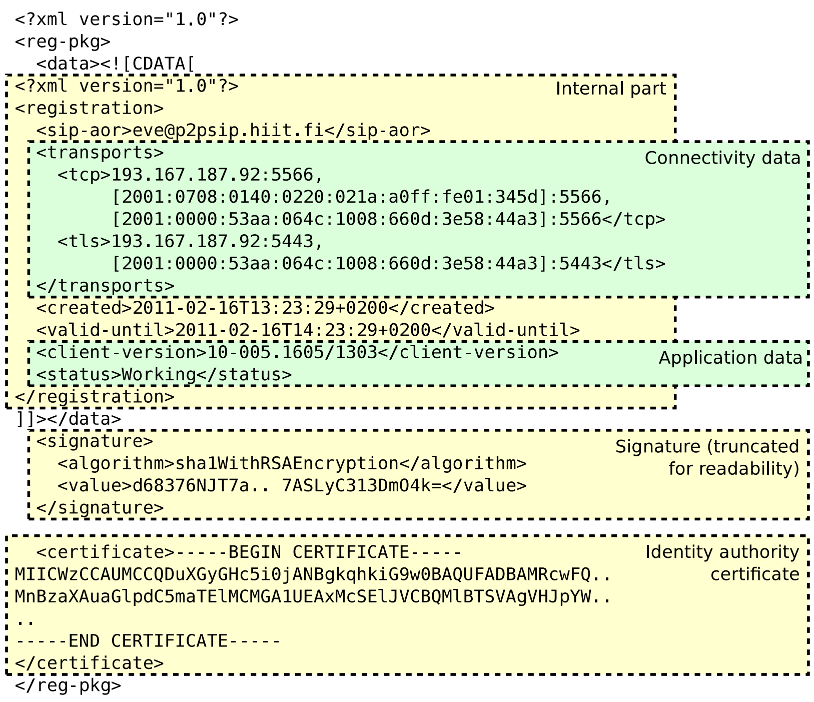
In addition to these, the prototype contains a set of utility modules, and application-specific modules through which the applications interact. These are explained in more detail in Chapter 3.3.

3.2.2 Overlay use

For peer lookup and connection establishment, we use a system of *registration packets*. The registration packets are electronically signed data containers that contain all the information needed to establish a direct connection, including current IP addresses, NAT traversal information (such as HIP RVS or Teredo addresses). They contain an internal and external part (as illustrated in Figure 3.5). The internal part contains the connectivity parameters as well as other, application-specific extensions. This data is signed with the identity key of the user (the publisher), with the signature, as well as any relevant identity certificates, contained in an outer *envelope*.

The internal part of the registration packet is meant to describe the current state of the user. Not only does it contain the connectivity-related parameters, but it can also include application-specific data such as the current status of the user, application version or other information relevant to a remote peer.

As these packets are signed, and relatively light-weight, the system is able to use any type of network storage that offers limited storage and a way of indexing data. This highlights one of the differences between our system and many other P2P systems. We do not stress building an optimal overlay network for locating users and distributing data, but are able to use a number of existing services for it, even simple web-sites. This approach is similar to the peer discovery mechanism of BitTorrent,



```
<?xml version="1.0"?>
<reg-pkg>
  <data><![CDATA[
<?xml version="1.0"?>
<registration>
  <sip-aor>eve@p2psip.hiit.fi</sip-aor>
  <transports>
    <tcp>193.167.187.92:5566,
         [2001:0708:0140:0220:021a:a0ff:fe01:345d]:5566,
         [2001:0000:53aa:064c:1008:660d:3e58:44a3]:5566</tcp>
    <tls>193.167.187.92:5443,
         [2001:0000:53aa:064c:1008:660d:3e58:44a3]:5443</tls>
  </transports>
  <created>2011-02-16T13:23:29+0200</created>
  <valid-until>2011-02-16T14:23:29+0200</valid-until>
  <client-version>10-005.1605/1303</client-version>
  <status>Working</status>
</registration>
]]></data>
  <signature>
    <algorithm>sha1WithRSAEncryption</algorithm>
    <value>d68376NJT7a.. 7ASLyC313DmO4k=</value>
  </signature>
  <certificate>-----BEGIN CERTIFICATE-----
MIICWzCCAUMCCQDuXGyGHc5i0jANBgkqhkiG9w0BAQUFADBAMRcwFQ..
MnBzaXAuaGlpdC5maTElMCMGA1UEAxMcSElJVCBQMlBTSVAgVHJpYW..
..
-----END CERTIFICATE-----
</certificate>
</reg-pkg>
```

Internal part
Connectivity data
Application data
Signature (truncated for readability)
Identity authority certificate

Figure 3.5. Contents and structure of the registration packets. In addition to connectivity information, the internal part may contain application-specific parameters.

where *trackers* offer a simple interface for storing, and fetching, peer information. Furthermore, we can use redundancy to create a sufficiently reliable system, even if the individual storage services are unreliable or untrusted. This eliminates the need to deploy custom servers to bootstrap the system, making the system accessible for ordinary users that do not have the necessary resources or know-how to maintain the infrastructure.

Prime examples of these open, free, storage infrastructures are cloud-computing services such as Google's AppEngine[3] and Heroku[4]. These allow users to deploy web applications, and provide different types of storage API with database-like features. Using these cloud services, a suitable storage back-end can be created, as we have done, with little effort. Other possible infrastructures include Amazon's SimpleDB service[5], which offers similar free quota-limited services. But even simple file storage services such as DropBox[6] can be used, as file names can easily be

[3]http://www.appengine.com
[4]http://www.heroku.com
[5]http://aws.amazon.com/simpledb/
[6]http://www.dropbox.com/

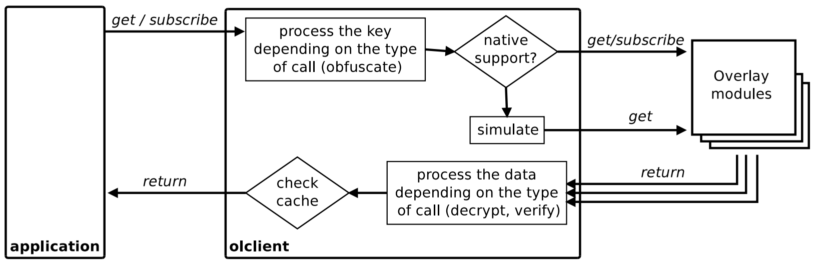

get / subscribe
process the key depending on the type of call (obfuscate)
native support?
get/subscribe
Overlay modules
simulate
get
return
return
check cache
process the data depending on the type of call (decrypt, verify)
application
olclient

Figure 3.6. Sequence diagram over how the olclient manages the overlay modules during get and subscribe calls. The key and return data are processed according to the type of call (obfuscation, encryption) fore issuing the request or returning the data.

used as indexes.

The management of the different types of overlay infrastructures (storage and routing services) is contained in so called *overlay modules* within the olclient. These modules are *registered* with the olclient manager either at start up or dynamically. The olclient manager controls these modules and delegates the calls appropriately. For instance, when retrieving an item, the olclient forwards the request to the appropriate functions of all of the registered overlay modules. As certain functions, such as retrieving data items signed by a specific user, may not be supported by all modules, the olclient manager simulates the call by examining and filtering the content retrieved using unsigned calls. The process is illustrated in Figure 3.6.

3.3 Application integration

An important part of validating our P2P model was to be able to use real use-cases and situations to evaluate it. The prototype model was designed to be flexible, supporting different types of data models and communication patterns. But as the prototype was implemented as a network proxy, it set some limitations on the types of applications we could support without having to modify the applications.

The first type of applications we considered was real-time communication applications. These are in many ways ideal for P2P communication systems, as they are based on exchanging data produced by the end-users. These are also highly personal applications, as we use them to communicate with people we have a relationship with. SIP was chosen as the primary protocol for real-time communication applications. SIP is an open, well documented protocol suitable for a wide range of different applications, including audio- and video calls, instant messaging and file transfers. SIP is also easy to work with, supporting network proxies, and a convenient messaging syntax. It also enjoys a wide acceptance and deployment, with numerous implementations.

Secondly we focused on creating an interface for HTTP-based applications. Although HTTP was originally created for the World Wide Web (WWW), it has been adopted by a wide range of applications as a generic communication protocol. Even though thought of as a client-server protocol, it is not bound to centralized management, making it suitable for P2P traffic as well. In the following chapters, we present the details of how these protocols were integrated. We conclude with a discussion on how the prototype relates to a more *generic* framework.

3.3.1 SIP

The SIP protocol adapted well to the P2P model, as it uses similar user identifiers as our framework, and is designed to support network proxies and processing of the signaling by these. As explained in Chapter 2.4, SIP supports distributed environments, but uses centralized, domain- based, authentication and lookup. These we replaced with the framework's own identity scheme and lookup, mapping the SIP identifiers (AORs) directly to the framework's. Applications (such as a VoIP application) could therefore be switched to our framework by simply configuring them to use our daemon as proxy for all signaling.

This SIP module accepts and interprets SIP signaling, parsing and modifying certain parameters (such as routing queues) to fit our network structure. Signaling destined for SIP servers (such as status updates and registrations) are captured and responded to appropriately. Signaling destined for remote peers (such as VoIP calls), results in a P2P connection to be established using the framework's lookup mechanism, and the message being forwarded. Our implementation supports all standard signaling, as well as the most common extensions to the protocol. Beside audio- or video calls, these include instant messaging and presence.

As security is one of the main topics of our research, the SIP module contains a multi-layered filtering system, with interfaces for examining and modifying both locally and remotely received signaling. Using these interfaces, we have added validity checks for the signaling, black- and white list filtering, and filtering based on social networks (as presented in Chapter 7). We also modify the session parameters in order to ensure that the data (such as the audio stream during a phone call) is routed securely through the encrypted P2P connection. Throughout this processing, the prototype will still behave as a normal SIP proxy, responding appropriately to the SIP application without the application being aware of it.

3.3.2 HTTP

The HTTP support was implemented to enable P2P exchanges of not only web- based content, but any type HTTP- based data, such as RSS updates and remote procedure calls (RPCs) using REST or SOAP interfaces. As with SIP, the goal was that users could use existing HTTP-based clients to access these P2P provided services, and standard HTTP servers for providing them. This required creating an URL mapping scheme, a service registration interface and to include authentication data to the HTTP headers.

In order to support legacy applications, we needed to preserve the URL syntax of HTTP, and use a mapping mechanism to convert host names to the identifiers used by the P2P framework. As explained in Chapter 3.2.1, the prototype uses service identifiers to refer to the applications of, or services offered by, a peer. We simply mapped the host name of an URL to the peer name, and the port number to these service identifiers. However, as the *at* character (`@`) of the peer names has a special meaning in HTTP, it is replaced with the sequence `.at.`. For instance, the URL

`http://alice.at.example.com:1000/get_rss`

would result in a request to a service registered as service 1000 on user alice@example.com.

The HTTP application handler was implemented as both a standard HTTP proxy as well as a local HTTP server using URL rewriting. As with SIP, the protocol handler parses the requests and establishes a peer connection in order to forward the request. In order for the recipient to be able to forward the request to the right HTTP- based server, we added a service registration interface. This REST- based interface allows applications to reserve a service number of the local identity by providing a local TCP port where the requests should be forwarded.

As all P2P HTTP requests will be routed through the local P2P proxy, they will all seem to be originated from the local host. Although these requests go through a similar filtering process as SIP, and there are multiple ways of adding authentication to HTTP, we also include the identity of the remote peer to each request. The HTTP headers of each request is rewritten, adding custom `X-P2P-From` and `X-P2P-To` headers specifying the source (remote) and target (local) peer identifiers. Although legacy applications may not use them, new ones can benefit from this automatic authentication and single sign on mechanism.

3.3.3 Towards a generic framework

It might seem contradictory to speak of a *generic* application framework, as the application protocols we chose to support (SIP and HTTP) each required separate adapters that both leveraged protocol- specific features. Supporting additional applications would require analyzing the protocol, in order to adapt it to the new environment. However, the SIP and HTTP protocols represent two important types of communication patterns, to which most other application protocols adhere.

As we move towards new network architectures and applications, we may need to create new, or improve old, application protocols to better fit those environments. However, the purpose of the prototype is not to show how any application can be integrated, but that most any *type* of application can. The SIP and HTTP protocols were merely two convenient, and well-suited, protocols for our work.

3.4 Summary

The goal of our framework has been to design a platform that enables experiments with networking applications in a distributed environment to study the issues they face. As the nature of the Internet is evolving beyond what it is designed for, there has been an active debate on new architectures. Many of the proposals put forth [96][168] show a trend towards models that favor more decentralized architectures. In these we have to rethink how applications operate, when there is limited access to established authorities. Although the architectures themselves have received a lot of attention, the impact these have on the design and operation of applications is often ignored. We have designed and implemented a framework that emulate these environments, in order to study the security and usability issues of the applications.

Secondly, our framework is designed to be usable today, easily deployed in both current and future networks. The security is based on strong identities, which also allows for features such as mobility. The underlying theme in our model is to tie these identities more closely to the application we use, and therefore to how we interact. By having a solid foundation for identifying not only hosts, but the users behind application instances, and even individual data, we can gain a more profound view of our network environment. This can be used to create a much richer context around the information we handle, enabling us to make better decisions. Our framework presents thus a more user-oriented environment for applications than the current host- (and network-) oriented one of today.

3.4.1 Technological advances

The implementation relies heavily on the HIP protocol and strong identities to provide connectivity and security. Although these components are not novel, the fact that we demonstrate how the model enables providerless overlays that can be deployed easily, and cost-efficiently, binding cryptographic identities to application instances to enhance security and control, we feel is of value. In short, the advantages of our model, compared to the current state of the art are:

I A cross-application identity model that authenticates users not only on a host level, but can even be used to control single application instances. Compared to existing models, this provides more accurate identification, as the user is better able to control which applications are able to use his credentials. Remote peers, and even network middleboxes, also have more control. For instance, the model enables remote peers to block, or whitelist, network access for a single application, instead of applying the same policy to all traffic from the host.

II Use of generic networking resources (such as on-line storage and cloud services) to create a platform for different types of applications and services. Currently new services require support from dedicated network resources. We demonstrate, through prototype validation and performance measurements, that by combining a HIP-based identity management and overlay with a generic lookup mechanism, it is feasible to construct private, and public, P2P overlay networks.

3.4.2 Related work

Our framework shares some aspects with existing P2P frameworks and applications. However, as it is designed as a P2P framework, not bound to any single application, it does differ substantially from most. For instance, content sharing systems, arguably the most prominent use of P2P today, operate with such different prerequisites and goals, that only certain details can be compared. These systems are optimized for locating and acquiring content without much care for the source, while we concentrate on the interaction and relationships between the nodes.

It does share some architectural similarities with BitTorrent, as noted in Chapter 3.1.2. Our overlay infrastructures serve a similar function, and operate in a similar manner, as the trackers of BitTorrent. As with BitTorrent, we require only a simple interface to be provided by these resources, while the implementation of those are left to the maintainer. The end-user can participate, as in the BitTorrent DHT overlays, but is not required. Furthermore, the system is not bound to a specific set of these network resources, and does not have to maintain a connection after suitable peers have been located. This eases the deployment of the system, and adds resilience, as it can use multiple simultaneously. Although the incentive mechanism is often quoted as the reason for BitTorrent's success, the minimalistic requirements for deployment has made it very approachable and easily adaptable.

Of the more generic systems, the RELOAD protocol (discussed in Chapter 2.5.2) comes, in many ways, the closest. Similar to our framework, it provides P2P networking for applications that provide interaction between users, but without being tied to a specific type of application. Compared to content sharing systems, the purpose is to establish links between the applications, not only fetch content. In these systems identity management becomes important, and RELOAD has opted a for public key- based scheme similar to ours. This enables secure authentication and for verifying the authenticity of content published in the overlay through electronic signatures.

However, REALOAD targets more established and stable environments than we do. Although it has a very modular architecture for supporting different types of overlay management, it does require that peers participate, and features even dedicated trusted nodes, the *enrollment servers*, for assigning responsibility. The network itself is relied on to perform certain tasks, such as access control, while our framework does not place any trust in the overlay infrastructures. Furthermore, even though RELOAD is not bound to a specific application, it does serve only a single one at a time. An identity is bound not only to a user, but also a specific application. Our framework unifies all interaction from a user under a single identity. This enables us to easier form relationships with our peers, as we can interact in different ways while being confident of the other's identity.

4. Evaluation

During our research, we have applied our prototype to different environments and uses, recording the implementation's behavior and performance. To demonstrate the feasibility of our model, we present the most essential of these evaluations. First, we examine the networking performance. This is important for understanding how the technologies discussed impact the responsiveness and overall experience of the applications. We continue by evaluating the prototype on a mobile device, showing how well the proposed framework performs, and the resource requirements in terms of CPU and memory. Lastly we present the results from our experiments with different overlay infrastructures, quantifying the resource usage and requirements for deploying a network.

By examining the framework from these angles, we believe that the evaluation provides a good understanding of the feasibility of our model. Not only in terms of the hardware that we can expect to support it, but also how easy it would be to deploy and the user experience.

4.1 Networking performance

The goal of the network performance evaluation was to determine whether the technological solution we have outlined can deliver a satisfactory user experience for ordinary users. Three configurations were used. First utilizing HIP for both NAT traversal and security. Secondly using only Teredo for NAT traversal. Finally we evaluated a combination of Teredo and HIP. In this configuration the data was secured using HIP, but encapsulated for NAT traversal using Teredo. Although using the native NAT traversal extensions of HIP is surely more efficient than combining the two protocols, it can require RVS servers that are not as widely deployed as Teredo. This makes the hybrid approach an interesting alternative for secure, and NAT-traversing, connections. Three characteristics were measured; the initial connection set-up time, latency and the effect of the protocols' overhead on bandwidth.

4.1.1 Test set-up

The measurements were performed using the P2P HTTP interface of the prototype. The client peer issued a request for pre-configured resources on the serving peer, and the duration of the different faces of the request processing were recorded separately. The resources were served by using the Apache HTTPD server that had been configured (registered) to accept connections through the P2P prototype. The P2P HTTP client was run on a Linux-based 2.2 GHz Intel Core 2 Duo desktop computer with 2 GB of RAM, and the P2P HTTP server on a 1.4 GHz Intel Core 2 Duo laptop with 4 GB of RAM. The computers were running the Debian GNU/Linux 4.0 operating system with a standard 2.6.28 version of the Linux kernel. The HIP for Linux (HIPL) version 1.0.4 [68] stack was used for HIP and Miredo 1.1.5 [118] for Teredo connectivity.

The overlay infrastructure we used for lookup was a simple Google AppEngine storage application consisting roughly of 20 lines of code. This application offers an HTTP interface for storing and retrieving arbitrary blocks of data using 160 bit keys. Similar to the interface offered by OpenDHT [127], each item has an expiration date and can be protected from unauthorized removal by a password. The default server for Miredo, teredo.remlab.net was used for Teredo, and HIIT's public RVS, ashenvale.infrahip.net, as the HIP RVS server.

As our framework is designed to enable new services to be deployed easily, without additional infrastructure investments, the main target group are ordinary home users. The evaluation was therefore designed to measure the performance of the framework in networking environments that ordinary, residential, users are faced with. Although this does introduce more uncontrollable interfering factors than when done in a controlled setting, the results describe better the performance of the framework from the user's point of view. With this in mind, the network performance evaluation was designed with the serving host behind a NAT on a residential digital subscriber line (DSL) broadband connection. The client host was placed 16 network hops away, on a different autonomous system. This provided a realistic scenario for what can be expected in terms of network quality for the potential users. The average round-trip times (RTT)

Table 4.1. Round trip time (RTT) between network elements.

Source host	**Target**	**RTT**
Serving	Client	194.2ms σ 0.7
Serving	Lookup	37.6ms σ 0.2
Serving	Teredo server	186.9ms σ 0.1
Serving	HIP RVS	194.0ms σ 0.5
Client	Lookup server	26.7ms σ 0.1
Client	Teredo server	50.3ms σ 1.4
Client	HIP RVS	1.0ms σ 0.0

between the network elements is presented in Table 4.1 to illustrate the quality of the connections.

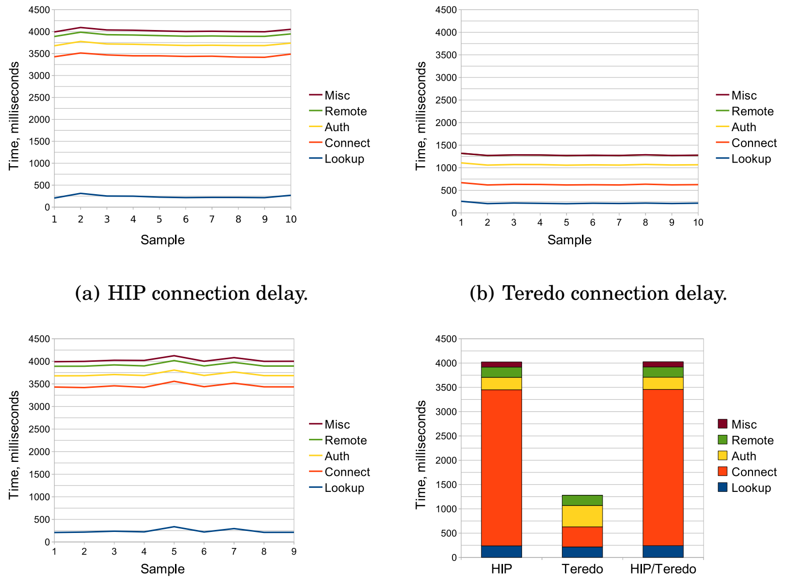
4.1.2 Connection delay

The connection delay is the time required to establish a P2P connection and respond to a request, illustrating how quickly users are able to connect to new peers. This delay was recorded using an internal logging mechanism of the prototype which automatically recorded the duration of each of the four main phases of the connection establishment. These are the *Lookup*, the lookup of the registration packet for the remote peer. *Connect* is the time needed to establish the actual P2P connection. *Auth* is the duration of the handshake, during which the identity of the remote peer is verified. *Remote* is the time the remote peer processes and responds to the request. Finally *Misc* contains the sum of all additional processing needed to establish the connection. This includes possible DNS lookups and various internal processing.

As a result of the NAT traversal, the initial connection establishment is more time consuming than subsequent requests. Both Teredo and HIP use UDP hole-punching techniques, which are based on UDP encapsulation and relaying of the first packet through an intermediate (Teredo server or HIP RVS). Figures 4.1(b)-4.1(d) show the times recorded for ten requests for each one of the three configurations. Between each request, both peers were reset to provide a fresh setting.

Teredo performs as expected, adding only the RTTs needed to involve the Teredo server to the connection process, resulting in an average of

1281 ms total. The HIP-based connections performed worse. The HIP connection establishment adds an additional 3000 ms to the process, with the total averaging 4026 ms. Only a small fraction (100 ms in *Misc*) can be explained by additional chores related to HIP (such as mapping HITs to IP addresses). As this delay is present both when relaying through the RVS and Teredo server, the HIP NAT traversal cannot be blamed either.



(a) HIP connection delay.

(b) Teredo connection delay.

(c) HIP / Teredo connection delay.

(d) Connection delay averages.

Figure 4.1. End-to-end connection delay in residential networks when using different transport protocols. Ten samples were measured for each configuration. The duration of each of the phases of the connection establishment are stacked to form the total.

After performing additional experiments and researching the problem, it was found to be due to the Linux kernel dropping the first packet of the IPSec ESP BEET tunnels created by HIPL. By default, HIPL uses the kernel-based IPSec, setting up appropriate routing tables for packets addressed with HITs. These result in the HIPL daemon being activated to perform the BEX when a packet destined for a new HIT is received. The first packet is subsequently dropped by the kernel while waiting for the BEX to complete. As the system uses TCP for the P2P proxy connections, the connection is completed only after the TCP retransmission timer expires, resulting in a new packet being sent.

Although this issue may be addressed in future versions of the HIPL implementation, it should be noted that this affects only the initial connection set-up, not subsequent requests.

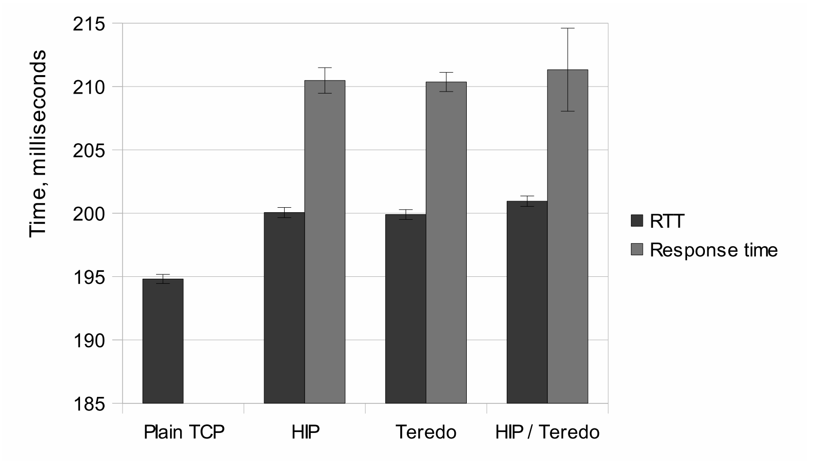


Figure 4.2. Average RTT and HTTP response times for each connection type. Plain TCP HTTP connections were not possible as the serving host was behind a NAT.

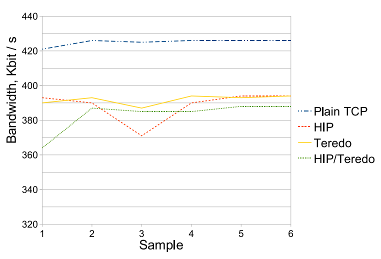
4.1.3 Latency

The latency was measured using pre-established connections. This provides a view of what the end-user would experience after the initial connection establishment, in terms of the responsiveness of the services. Figure 4.2 depicts the average RTT measured using ICMP ping packets for each connection type, as well as the corresponding response times for the actual HTTP content. The HTTP response time was measured by issuing a request for a small (50 bytes) web page. The time recorded is the time between the client application issuing the request and when it actually had received the content.

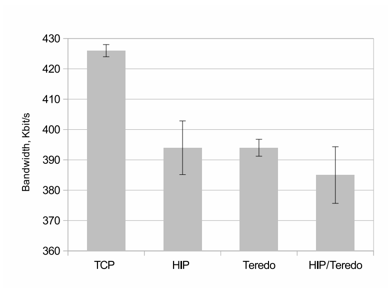
The figures are very similar between the different connection techniques. After the initial set-up, each protocol sends the packets directly between the hosts, adding only protocol encapsulation. The slightly longer times for the HIP/Teredo combination connections can be explained by the additional encapsulation and processing.

4.1.4 Throughput

The throughput was measured as the maximum rate at which the serving host is able to deliver data over a TCP connection. The measurements were done using the iperf network performance measurement tool. Figures 4.3(a)-4.3(b) show the results when using the three different connection techniques as well as without any encapsulation (plain TCP) as reference. Each measurement was run for a duration of 20 seconds to



(a) Throughput, per sample.



(b) Throughput, average.

Figure 4.3. Throughput measured in Kbit/s when using different connection types.

obtain a stable result.

It should be noted, as seen in the graphs, that these are highly susceptible to external influences (other users, link quality and the Internet Service Provider, ISP, policies), but still provide an insight to how these protocols affect the performance. As the figure shows, the encapsulation does have a noticeable effect on the overall bandwidth. Both HIP and Teredo decreased the maximum bandwidth from 426 Kbit per second to 394 Kbit per second, a decrease of 8%. Perhaps surprisingly, the HIP/Teredo combination resulted only in an additional 1.5% decrease from that value, with a maximum bandwidth of 388 Kbit per second.

Figure 4.4 compares the total IP packet size generated by a 238 byte HTTP request using the different connection types. As the figure shows, Teredo and HIP add approximately an equal amount of overhead to the data packets (40 and 42 bytes in this example, although the ESP header length may vary due to padding). This corresponds quite well to the throughput measurements. It should be noted that although the Teredo/HIP combination adds the most overhead, it is only 42 bytes more per packet. Although this is roughly a 33% increase in the total amount of overhead compared to plain Teredo, it is below three percent of a typical network MTU of 1500 bytes.

In this example, our prototype is responsible for generating roughly 30 bytes per HTTP request of additional data (modifications made to the original HTTP request, packet type- and stream-management identifiers). This is a relatively high figure, and can surely be optimized in future revisions.

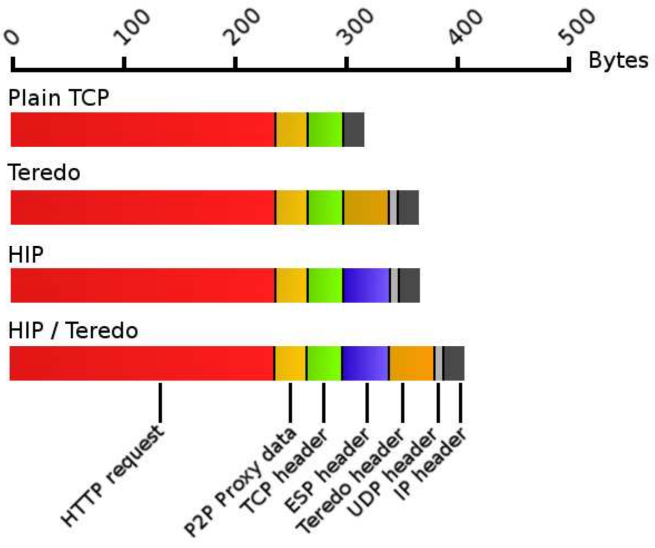


Figure 4.4. Comparison and content of a packet containing 238 bytes of application data produced by the different protocols.

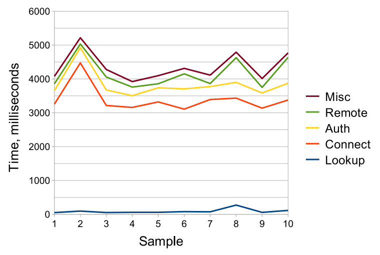
4.2 Performance on a mobile device

For evaluating the framework's performance on mobile devices, we used Nokia Internet tablets (N800 and N810), as they feature a suitable OS for the prototype and currently represent lower-end smart phones in terms of hardware specifications (which highlights any issue we face).

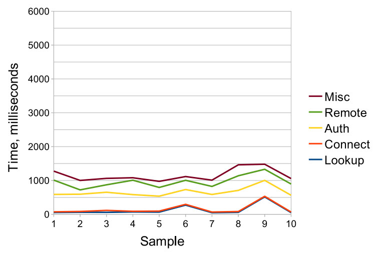
We used the built-in Internet call and instant messaging application for conducting the evaluation. This application supports the SIP protocol, and can therefore be used together with the P2P framework without modifications. During our evaluation, we used the devices in two different network environments. Connectivity is provided through a wireless LAN (WLAN) interface, which we used both in managed (connected to a WLAN access point), and ad-hoc modes. This provided a view of how the application performs when connected to a global network, as well as the usability in closed networks.

The most notable issue was the latency created by the HIP base exchange. As noted by Khurri et. al. [94], the throughput of IPSec on these devices is sufficient for the data of multimedia sessions, but the delay caused by the base exchange is considerable. As our prototype provides no indications of the state of the HIP stack, it can lead to frustration as the user is unsure whether the call is progressing. However, this affects only the initial connection set-up. Subsequent requests are not affected, and are delivered with the latency comparable to that measured in Chapter 4.1.3.

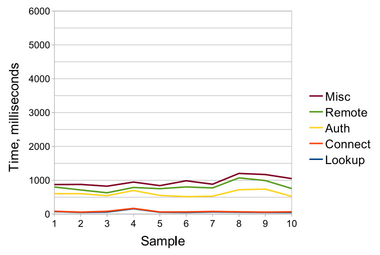
4.2.1 Connection delay



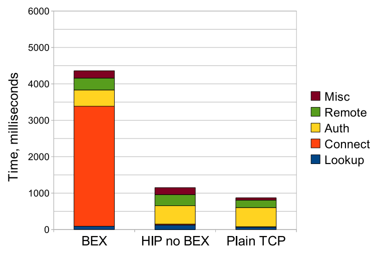
(a) HIP connection delay.



(b) HIP connection delay after initial BEX.



(c) Connection delay using plain TCP.



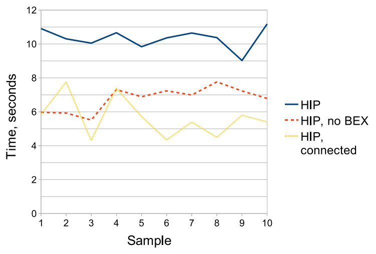
(d) Connection delay averages.

Figure 4.5. Measured connection delay on Nokia N810 with and without HIP.

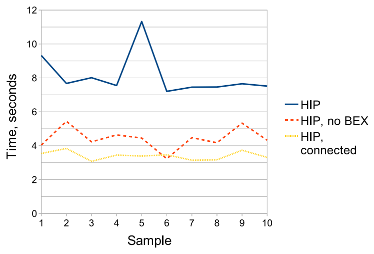
The connection delays were measured using two Nokia N810 Internet tablets connected through an ad-hoc WLAN network. The tablets were situated near by, to minimize possible network interference. This allowed us to concentrate on the performance of the tablets, as the impact of the network has already been evaluated previously. Local network UDP broadcast was used for the lookup. We used HIP for the P2P connections, but also compared with using plain TCP to highlight the affect the strong security has on these devices.

The connection delay was recorded by the same internal logging mechanism, presented in Chapter 4.1.2, over ten calls. Three sets of measurements were taken. First using HIP for the P2P, and resetting both devices between each call. This illustrates the delay of the initial request (or call) between the mobile devices. Secondly we measured the delay when a HIP connection had been pre-established, illustrating the delay of subsequent requests between the two. Finally we measured ten calls when the P2P connection is established without HIP.

Figures 4.5(a)-4.5(d) show the time recorded for each of the three sets. Using HIP on these mobile devices increases the initial connection delay substantially. However, after a connection has been established, HIP affects the performance only slightly.

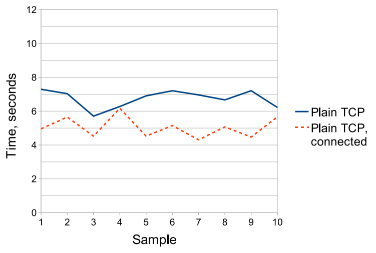


(a) VoIP application inactive.

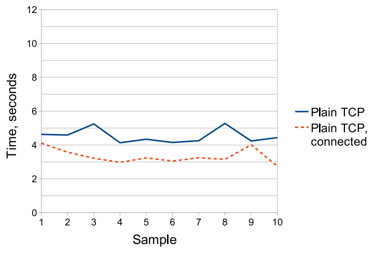


(b) VoIP application active.

Figure 4.6. Call set-up delay with HIP.



(a) VoIP application inactive.



(b) VoIP application active.

Figure 4.7. Call set-up delay without HIP.

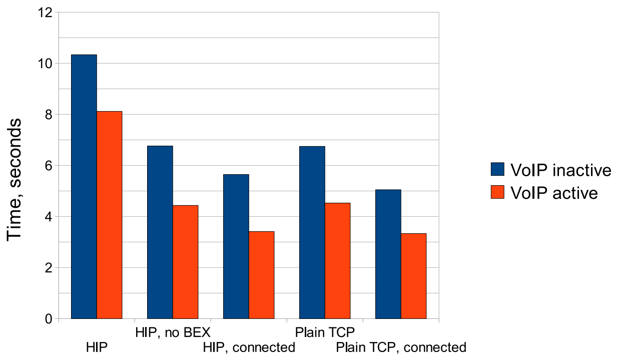


Figure 4.8. Call set-up delay averages.

In order to understand how the overall experience is impacted by these delays, we measured also the overall call set-up delay. This is the total delay a user experiences, which includes both the delay caused by the networking stack, as well as the time required by the VoIP application for processing the signaling and updating the UI. In addition to the three configurations used for the connection delay (HIP, pre-established HIP and plain TCP), the call delay was also measured when a P2P connection had been pre-established (subsequent calls to the same peer). Furthermore, the call delay was measured both while the receiver was initially idle, as well as when the VoIP application had been pre-loaded.

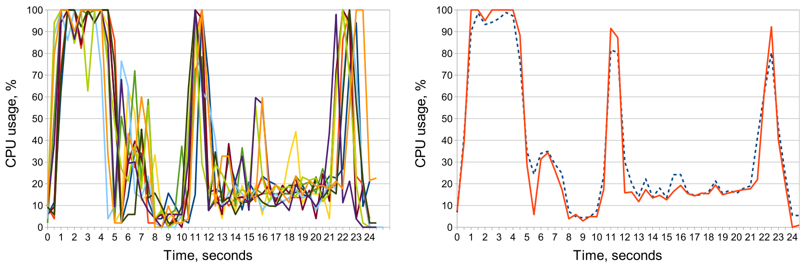
Figures 4.6-4.8 show the times measured for each of these configurations. Although 4.5 seconds for establishing the HIP connection (Figure 4.5(d)) seems long, the additional delay caused by the built-in VoIP application is much longer. As Figure 4.8 shows, merely starting the application consumes approximately two seconds, leading up to an over ten second average delay when placing a call to a remote user without a pre-established connection.

4.2.2 CPU usage

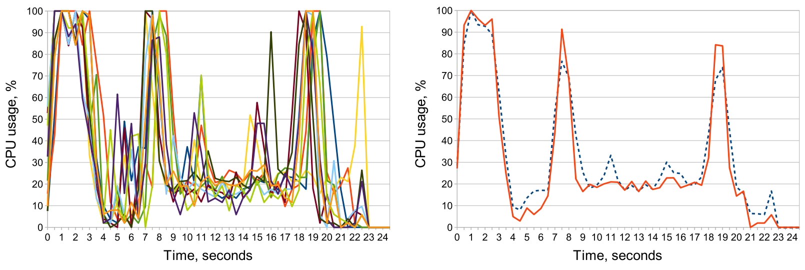
As the results presented in Chapter4.2.1 showed, although the performance of the prototype on the mobile N810 tablets may seem poor, it is not excessively worse than the overall application experience of those devices. The Nokia N810, released in 2007, features a 400 MHz single core ARM processor (TI OMAP 2420) and 128 MB of RAM. With the rapid development pace of electronics, this represents a rather low-end mobile device today, when modern smartphones feature multiple core processors, efficient 3D graphics capabilities and giga bytes of RAM. However, by measuring the CPU usage on the Internet tablets, we can analyze how the lack of processing power affects the framework. This can be used as an indicator for how future, more efficient, devices may cope with the system.

We measured the CPU usage of the Nokia N810 tablets during the connection delay measurements presented in Chapter 4.2.1. The measurements were done by polling the Linux CPU usage counter (available through the $/proc/stats$ device) at 0.5 seconds intervals. This provides data on what types of tasks the processor has been occupied with during each of the 0.5 second intervals. The CPU utilization can be calculated by subtracting the different tasks (such as kernel and user processes, interrupt request processing) from the total.

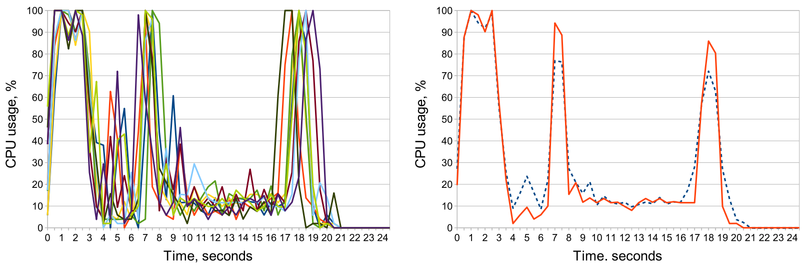
Figures 4.9(a)-4.9(c) shows the CPU utilization measured when placing the calls in the three different configurations. The measurements have been adjusted as close as possible to start at time zero for each iteration, and depict identical sessions where the recipient answers the call and keeps it open for 10 seconds. The left-hand side graphs show all of the individual measurements plotted in the same graph, while the right-hand graphs show the averages of these, as well as a typical instance (the median for each point).



(a) HIP, including initial BEX handshake.



(b) HIP, with pre-established connection (no BEX).



(c) Without HIP

Figure 4.9. The CPU utilization on a Nokia N810 Internet tablet when making a 10 second long P2P VoIP call. The right-hand graph shows all 10 measurements, while the left-hand the average (dotted line) and median (continuous line).

A clear pattern can be seen in each on of the graphs. When initiating the call, the CPU utilization rises sharply to 100%, as the VoIP application initializes itself and the P2P proxy establishes the peer connection. As the signaling is transmitted to the remote user, the CPU utilization drops, as it waits for a reply. At 6-10 seconds into the measurement, depending on the connection establishment type, it rises again as the response is received and the call connected. It continues relatively low (at 10-20%) as the call is in progress, but rises again as the call is disconnected.

Although the initial spike in CPU utilization lasts longer for when no pre-established HIP association exists (Figure 4.9(a)), it is only one or two seconds more than in the other cases. Furthermore, the spikes when the call is finally connected and disconnected have little to do with the P2P prototype, as these do not require the prototype to perform additional processing. We can conclude that the P2P prototype itself is not as restricted by the low processing power of the Nokia N810 as its own built-in VoIP application, which was partly illustrated by the call delay measurements as well. In addition, the use of HIP for the data connection does not seem to be an obstacle, as the CPU usage remains at approximately 20% during the call (compared to 10% when not using HIP).

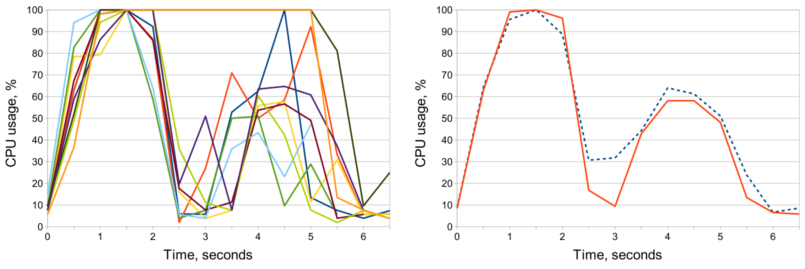
To measure the CPU usage of only the P2P prototype during calls more accurately, we repeated the measurements using the P2P HTTP interface of the proxy. This removes the load caused by the VoIP application, highlighting the processing characteristics of the P2P prototype. The measurements were done using the wget command-line utility, adding a minimal amount of additional load. As we wanted to measure only the CPU usage during connection establishment, the recipient peer did not have a HTTP server responding, but returned an error instead.

Figures 4.10(a)-4.10(c) shows the results of ten measurements for each connection type. As the graphs show, the BEX of the HIP connection establishment consumes significantly more processing power than the rest of the exchange. Again, the use of HIP for the data connection is not an issue, as the CPU usage after the BEX (Figure 4.10(b)) is only slightly higher, on average, compared to the non-HIP (Figure 4.10(c)) connection.

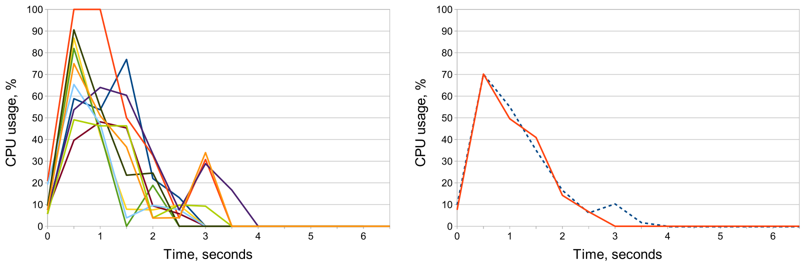
4.3 Load on storage

As our framework establishes direct P2P connections using information (the registration packets) stored in the overlay infrastructure, the load on these storage service presents the main factor affecting the scalability of the system. Increasing the number of users does not increase the load on individual peers, or affect the connections between these, only the load on the storage service. The number of peers the storage service can serve, and how well, limits thus how large a population is able to use the system.

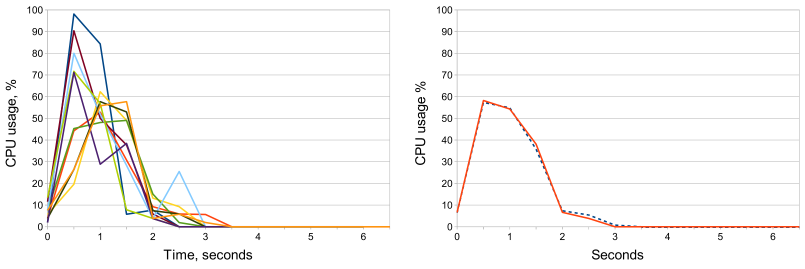
The overlay infrastructures, as explained in Chapter 3.1.2, need only to provide a simple key-based data storage service. In our prototype, we im-



(a) HIP, including initial BEX handshake.



(b) HIP, with pre-established connection (no BEX).



(c) Without HIP

Figure 4.10. The CPU utilization on a Nokia N810 Internet tablet when making a P2P connection. The right-hand graph shows all 10 measurements, while the left-hand the average (dotted line) and median (continuous line).

plemented support for a simple HTTP- based interface containing three key- based operations: get, set and remove. During our evaluation, we monitored the network traffic and recorded that storing a registration packet generated roughly 3.2 kilo bytes (KB) of upstream, and fewer than 200 bytes of downstream data in total when using this HTTP-based storage interface. A single lookup resulted in approximately the same amount of data, although in reverse directions. Storing a packet required also roughly 3.2 KB of storage space.

Considering a system of a thousand users updating their status once every five minutes (as was the default in our prototype), maintaining the system would generate approximately 900 MB of upstream and 56 MB of downstream traffic in 24 hours, with 3.1 MB stored in the database at any time (assuming old packets are replaced when updated). Using the figures from Facebook[1], arguably the most popular social networking site today, the average number of *friends* per account was 190 as of May 2011 [173]. Assuming a similar social network, and that each user would contact, on average, each one of his contacts once a day, it would add another 37.1 MB of upstream and 593.8 MB of downstream traffic. This interaction would total in 937.1 MB upstream and 650 MB downstream traffic per 24 hours, with 3.1 MB of data stored at any given time.

To better understand the scalability and cost efficiency of the framework, we examined the pricing policies of Google's AppEngine and Heroku, two commercial platform as a service (PAAS) providers. Although there are other similiar cloud- hosting services, these are amongst the most popular, and offer representative pricing models.

Both Google's AppEngine and Heroku offer different pricing plans, including a limited free one, based on the amount of resources (storage, processing power and bandwidth) that are used by the application. Currently the AppEngine limits the free service to 1 GB of data transfer per 24 hours, and 1 GB of indexed storage per application[2]. Heroku offers more bandwidth (2 TB per month), but allows only 10'000 entries to be stored in a relational database in its free offering[3]. Considering our example of a thousand user population, both of the free plans offer enough resources to meet our demands in terms of bandwidth and storage.

Beyond the free offering, the two follow different pricing models. Google's AppEngine scales in a fine grained manner, dividing the resources into very specific components. For instance, inserting an entry to the database consumes 1-7 so called *write* operations, depending on whether an old entry is updated and the indexing used. Retrieving a value consumes 1-2 *read* operations and 0-1 *small* operations, depending also on the database set up. Each of these *operations* have separate quotas, and can, together with a wide array of other resources (such as storage space, bandwidth or CPU utilization), be increased independently in small increments.

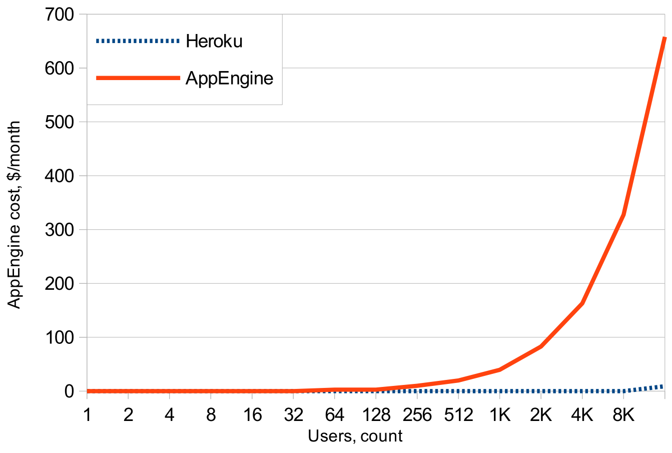
Heroku, on the other hand, offers fixed plans with specific performance guarantees that are purchased beforehand. Compared to Google's AppEngine, these plans are much simpler, concentrating on easily under-

[1]http://www.facebook.com

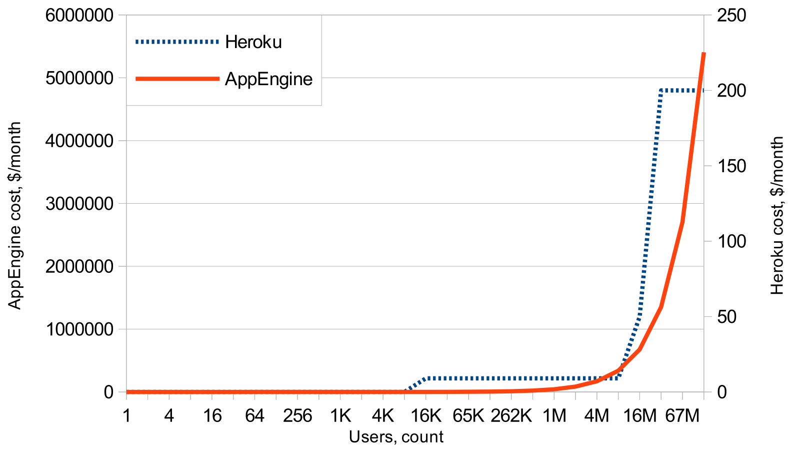
[2]https://developers.google.com/appengine/docs/quotas

[3]https://www.heroku.com/pricing

stood resources such as storage space, CPU utilization and add-on software. Extending our naive resource consumption model beyond one thousand users, we see in Figure 4.11(b) the total cost of hosting different sized networks on these two platforms, when considering expenses related to the bandwidth use and storage- related costs.



(a) Cost of hosting, small networks.



(b) Cost of hosting, large networks.

Figure 4.11. The cost, per month, of hosting different sized networks on Google AppEngine and Heroku. The first figure shows the cost for smaller networks, using the same scale for the cost of both systems. The second figure uses different scales due to the dramatic difference in cost.

As the figure shows, Heroku provides orders of magnitude less expensive hosting for our simple key-store application as the network size grows. For instance, hosting a one million peer network would results in bandwidth- and storage costs of over 42 thousand dollars per month on Google

AppEngine[4], while the same application on Heroku could suffice with a nine dollar plan (as of November 2013), mostly due to the looser bandwidth restrictions and unrestricted database access on Heroku. These calculations do not account for additional CPU resources required for processing the requests, which would increase the cost depending on the quality of the hosted application's implementation and the performance requirements.

In order to verify these assumptions, we implemented storage services offering the HTTP- based interface on these platforms, as well as a stand-alone version, and experimented with a small-scale simulation. We created an application mimicking the requests made by the clients, and deployed a 450 node network, simulating a medium sized community, using our department resources as well as the PlanetLab network. Both the Google AppEngine and Heroku applications were deployed using the free service plans of both. The stand-alone version was implemented as a Python application using an in-memory database, and deployed on a Linux-based 2.2 GHz Intel Core 2 Duo desktop computer with 2 GB of RAM on the same local network as the client nodes.

We ran the applications for five days for each of the three storage implementations. Each one of the client nodes made a registration request using actual registration packages (stored under unique keys) once per 10 minutes. Call set-up was simulated by having each application request 190 randomly chosen registration packets per 24 hours. We recorded the both the success rate of the registrations and call- set ups, as well as the average response times.

The results were in line with our assumptions. The stand-alone and Heroku- hosted applications performed well, with low latency and virtually no operation errors (the success rate for all requests were over 99.99%). The Google AppEngine displayed also good response times on all requests (indicating that the server never became overloaded), but the daily quota was quickly filled (on average within 59 minutes after the daily reset).

The quota that was first exceeded was the *Datastore Write Operations*, of which only 50 thousand operations is available per day in the free service plan. Following it was the *Datastore Small Operations*, recording

[4]The calculations were based on the data consumption of our prototype implementation. The price quotes represent therefore the worst case scenario, as a production-grade version would surely be more optimized.

database management operation such as row counts, whose quota was exceeded on average after 185 minutes. The third quota that got exceeded was the *Database Read Operations*, on average after 11 hours of the simulation. All other resource consumption of the Google AppEngine application remained within their limits.

The storage space consumption (bytes stored in the databases) remained far below the quotas on the commercially hosted services, and the bandwidth use on Google AppEngine was only 8% of the downstream limit and 14% of upstream[5]. Heroku does not provide values for the bandwidth utilization, but the fact that virtually all requests were completed successfully indicates that the use was within the limits of the free service plan.

4.4 Summary

The evaluation has shown that our prototype, and thus the model we have proposed, is feasible both for different types of network environments, and different types of devices. By using low-end mobile devices, we have demonstrated that even though we emphasize security, the model does not pose an unbearable burden on the devices when operating normally. Furthermore, by creating storage services using free cloud resources, we have shown that private overlays can be created using freely available network resources. However, the evaluation did highlight some of the weak points of the model.

The call set-up delays and CPU utilization measurements of Chapter 4.2 did show that although the performance of low-end mobile devices is sufficient for casual communication, the devices can not serve a larger crowd by themselves. The authentication procedure required for the strong security is resource-demanding, which can easily lead to mobile devices being exhausted connection requests. Malicious exploitation of this weakness can be subverted using different types of filtering (as discussed in Chapter 7). Application that require maintaining a large set of connection should however be hosted on more powerful nodes.

The storage evaluations showed that although it is possible to host overlays using only free resources, the nature of the limitations of these differ, which affect how they can be used. Our experience with the cloud-

[5]The lower than expected downstream utilization was attributed to Google AppEngine's use of HTTP transport compression.

provided storage services showed that even though a resource might at first seem suitable (in terms of bandwidth and storage quotas), there might be other issues that make them unusable for our model.

5. Privacy extensions

Privacy in P2P systems has been an actively discussed topic for a number of years, with research touching on a range of issues. Protecting users and their actions is difficult in P2P systems that rely on the close cooperation between nodes. Much of the work in this area has concentrated on hiding the real identities of users [120] [54] [111]. This has often been the most pressing concern in existing networks where users of content-sharing systems risk legal reprimands, or are threaten by political and social oppression when the networks are used to express ideas and opinions. As long as the users cannot be identified or traced, there is no need to hide the actions (such as content requests) unless the content itself contains something revealing.

As we are creating a generic application framework based on strong, persistent, identities, we need to re-examine the privacy issues. When the services are provided by the end-users, networking also becomes more personal, revealing not only what we do, but also with whom we interact, and the relationships we have.

Communication applications, such as Internet telephony, are prime examples of applications that are very personal in nature, and where the need for privacy is well understood. These have traditionally relied on a system of trusted, centralized, servers for authentication and setting up connections. The privacy of the users, with respect to the system operator, is non-existent. Although protective measures, such as phony accounts, source address hiding and encryption of the data streams can be used, the operator has all the means to track the call records and other communication made through the system. This may not be a concern for most consumers, since the operator is often seen as trusted; however, for companies and governments this raises more fundamental issues. For instance, many companies block the Skype communication application as its call routing protocol is proprietary which makes it impossible for to verify its security.

As described in Chapter 2.1, P2P systems form a network between the participating nodes, used to collectively perform tasks and manage resources. P2P communication systems, in particular, use this network to replace the centralized service provider, and relies on it everything from connecting the calls to managing presence, contact lists and other data.

In the model we are proposing, the role of the trusted service provider is taken by the overlay infrastructures. The control of these is shared amongst the peers, with the implication that anyone that is part of the network can track the communication made through it. As these operations, and data, are so inherently bound to the users involved (through electronic signatures), simply hiding the source of requests becomes irrelevant. Furthermore, as we are designing a system to be used with different types of overlay infrastructures, we cannot rely on being able to control who within those infrastructures are allowed to access our data. A more systematic privacy mechanism is needed.

In this chapter, we examine a simple application-level model for enhancing privacy in distributed communication systems based on persistent identities. We review the details of the threat in current systems, which we follow by presenting our solution, discussing its benefits, tradeoffs and possible alterations. As the model imposes restrictions on the accessibility of users, we continue by discussing issues related to the usability of the solution. Finally we present how the model was implemented, and compare it to existing privacy enhancement schemes for P2P systems.

5.1 Problem scope

As there are few public P2P systems based on strong, persistent, identities deployed today, there has not been much effort put into solving the privacy issues in these environments. The common approach has been to encrypt the data passed within the overlay, although it still leaves the signaling (between whom the data has passed) vulnerable. This is also the approach taken by the IETF P2PSIP working group [74], arguably the most prominent attempt at creating an open standard for these systems.

As described in Chapter 2.5, the P2PSIP protocol defines a highly modular framework supporting different applications (called *Usage*s) as well as overlay network types [78]. The network module of this framework offers message routing, key-based storage and connectivity services through a common interface, independent of the underlying network structure. Although there are few restrictions, the underlying network is assumed to be a distributed hash table (DHT)-like structured network, with efficient key-based routing, where the storage service scatters the data throughout the network.

The P2PSIP protocol defines three operations for the SIP Usage; registration, lookup and connection establishment, which correlates to our model of registration packet management and use of overlay infrastructures for connection establishment. Registration and lookup is based on managing data packets in the network under keys made from the hash of SIP AORs. Connections are established by completing an ICE procedure, where address candidates are exchanged by routing data packets over the network. These operations offer a number of opportunities for curious, or malicious, users to eavesdrop.

Assuming a DHT-like overlay (such as Chord), where responsibility for data is assigned using a key proximity function, any node with a suitable NodeId, along with every node in-between, is able to intercept registrations. Even if these nodes were trustworthy, it is trivial to request the information using the public SIP AOR. To establish a SIP session, the caller fetches the registration packet for the responder, and connects using the Attach function. Again, the node maintaining the registration packet, and all nodes in-between, can easily monitor from whom the user receives calls. And even though that path is secured, Attach-related messaging offers yet another opportunity to track the call.

As noted in the draft [78], end-to-end encryption of the payload could be used to mitigate some of these, although still leaving the storage keys and NodeIds exposed. Also, a strong authentication mechanism and a restricted identity acquirement procedure would prevent peers from positioning themselves in the network, decreasing the possibility for eavesdropping and sabotage in large networks [42]. Although our model rely on a different mechanism for connection establishment, we do face the similar privacy issues when publishing the registration packets, as these are publicly available in the overlay infrastructures under well-known keys.

The problem at hand is to introduce privacy mechanisms to P2P communication that prevent intermediate nodes from intercepting and tracking the communication through the overlay, such as call signaling, while maintaining strong, persistent, identities. We need a system that does not rely on cooperation from the overlay, but can be used with our model of untrusted, key-based, storage services. This results in a privacy mechanism that is portable and easily adopted to other systems as well. Although, as we will see, the mechanism may be simple, the key is to construct it so that it is efficient and usable, with the risks and potential trade-offs understood by the end-users.

5.2 Solution model

The aim was to design a simple model which sets no additional requirements on the underlying storage service, using only the get, put and remove primitives. In addition to making the scheme adaptable to other, similar systems, it does not prevent the use of additional privacy-related enhancements, such as source address hiding, for even greater security.

The strong, cryptography-based, identity scheme used by our framework plays a key role in protecting the privacy of users. It allows users to protect the integrity of their data and make it accessible only for specific peers using public key encryption. Similarly to the P2PSIP protocol, this is the first step of the privacy enhancement: to encrypt the signed registration packet using the public key of peers that with whom we may want to establish a session. We therefore *publish* this data only for a specific set of trusted peers.

By encrypting the registration packets, as well as any other data we store in the storage services, we ensure that only the intended recipient is able to read the details of that data. In the case of the registration packet, the user status, connectivity information and other application information is safe. But as the keys used for storing registration packets map directly to the public identities, encrypting the content of these packets alone does not prevent intermediate peers from tracking connections. Size analysis can be used to identify registration packets and by monitoring the keys used to store and request these, we can determine both who is available as well as the recipient of a call. Although buffering and *decoy* packets can be used to make it harder, it seems unavoidable that the storage keys should also be obfuscated.

5.2.1 Storage key obfuscation

So far, we have assumed that users need to possess only the public keys of the trusted peers in order to publish registration packets. We could consider a scheme where the storage key (the name of the publisher) is also encrypted using these. However, this is easily broken in systems were the public keys are well known, as in our model. An eavesdropper could simply re-construct this key, and continue monitoring who is present by following the updates made to the data.

We could protect the publisher by publishing the encrypted registration packets using a storage key tied to the recipient. For instance, if both Alice and Bob would like to share their presence with Eve, they would both publish their registration packets under the same key. As the key for this *index* is shared and the content encrypted, intermediate nodes cannot determine who the packets concern. This protects the publishers, but reveals information about the recipient, such as the number of friends and availability (a lookup reveals that Eve is online and probably about to call *someone*).

The most practical solution we found is to simply use shared secrets to obfuscate the storage keys. After encrypting the registration packet, the storage key is formed by combining the shared secret with the user identifier and computing a hash digest, similar to a password salting scheme. The key will appear random, revealing nothing about either the source or recipient of the packet.

5.2.2 Usability considerations

Besides the technical details, the scheme introduces a number of usability issues that need to be considered. The need for a privacy-enhancing scheme might be unclear for many users accustomed to traditional communication systems operated by a trusted provider. Also, how such a scheme affects the usability of the system, how certain contacts or services might become unreachable as a result, is not apparent.

When activated, we need to have both the public key and a shared secret with everyone that may wish to communicate with. Otherwise we, or any service we are providing, will appear unavailable, *offline*. This might make it seem as an extension of a presence scheme, allowing us to lay hidden from unwanted contacts, or signaling when we are available. This may cause confusion as there are already application-level controls for this. However, the purpose of the privacy scheme is not to limit who is able to contact you, but to hide who *does*. This forms a layer below presence, which in turn is used to filter visibility. Furthermore, although shared secrets are often associated with close relationships, the model encourages users to establish these with anyone they are in contact with. The aim is to affect only the visibility of the P2P sessions, not to define the access control policy.

As the scheme requires users to possess each other's public keys, as well as have established a shared secret prior to contact, bootstrapping is a problem. High-security systems and concerned users might go through the trouble of creating and distributing these manually. However, most users do not appreciate the limitations and extra work, and would be satisfied with a partial open exposure, at least at times, but with the privacy enhancements used whenever possible. The shared secrets could then be agreed on during the first contact, stored to be used for subsequent sessions. This means that users may apply the enhancements using different policies, and we will not know whether a peer is using them at any specific instance. The scheme becomes configurable, with options reflecting both the privacy requirements we have, as well as the ones we assume the remote peers do.

This leads us to the question of how to present these options, *modes*, to the user. Although related to the traditional concept of presence, it has a slightly different meaning and effect, not familiar from centralized systems. In the following chapter, we present how this is solved in our model, discussing also the potential issues with our solution.

5.3 Implementation

The implementation of the privacy extensions included creating a local peer database and modifying how the data in the distributed storage is managed. The prototype maintains a database of the public keys and shared secrets of all the peers it has been in contact with. Although it is possible to manually configure a shared secret, a simple key-negotiation protocol was implemented, which automatically establishes (or reconfigures) a shared secret when connected.

The privacy enhancements, affecting how the registration packets are retrieved and published, are activated from the settings of the prototype. After analyzing different use-cases, we came to the conclusion that three different modes are needed:

Open. The registration package is published using both the open and secret storage keys. Lookup is done using only the open key, even when a shared secret has been established with the recipient.

Relaxed. As in the Open mode, the registration package is published using both keys. The lookup differs; if a shared secret has been established, only the secret storage key is used, otherwise the open one.

Paranoid. When set to paranoid, only the secret keys are used, both when publishing and performing lookups.

The Paranoid mode is the most secure; the prototype uses the privacy enhancement fully, revealing as little as possible. But as discussed in Chapter 5.2.2, for the enhancements to be usable, we also need to have an option of being openly exposed, at least temporarily. The need for two, nearly identical, modes is based on the assumption that nodes might get reset at times for different reasons, losing the database of shared secrets. This would result in two peers being in incompatible states, as one would assume they share a secret while the other one does not.

The Relaxed mode operates using a *best-effort* principle. Whenever a shared secret has been established, the privacy extensions are used without even trying the open key. By falling back to the open key, we would avoid the additional Open mode. But this could be seen as a violation of the privacy of the recipient, if in Paranoid mode. By falling back on the use of open keys in the Relaxed mode, either by using subsequent or simultaneous lookups, we might reveal a relationship between the two. Furthermore, the usability also improves, as the user knows beforehand whether the privacy enhancements are used. The usability of our solution is examined in more detail in Chapter 8.

Although the privacy enhancements primarily affect the management of the registration packets, the overlay management was extended to support generic privacy-aware overlay use. These extensions to the olclient module allows applications to specify a recipient for any data published and the expected source for the data retrieved. Using this information, it is able to apply the same privacy policies used for the registration packets to all data. This allows us to share application data, such as off-line messages and web-cache indexes (discussed in Chapter 6) in a privacy-conscious manner.

5.4 Bootstrapping and configuration

The Relaxed mode is thought of as being the default scheme for most users. As a shared secret is negotiated during the initial session, it will provide a level of privacy without requiring pre-shared secrets and manual configuration. After establishing the shared secrets, users can switch to the Paranoid mode to prevent intermediate nodes from tracking their status (by preventing the publication of the open, unencrypted, registration packets).

These modes could also be implemented using warnings and a reachability setting. When attempting a connection to a contact without a shared secret, the system would warn the user, providing an option of canceling the request. The reachability setting would control whether the open registration packet is published; whether people with whom the user does not have a shared secret would be able to connect. However, these would require cooperation with the applications that use the framework (e.g., the VoIP applications) in order to provide a meaningful experience.

The Open mode is intended primarily as a fall-back, in case of a reset of the database of shared secrets, for instance after re-installing the software or when changing devices. After such events, it does require that both the user, and his contacts, at least temporarily switch to a lower (Relaxed or Open) scheme in order to be able to connect. Although the database would surely be saved, or synchronized, during these upgrades, establishing relationships with users using the Paranoid mode still presents an issue. Although the problem can be mitigated using various introduction- mechanisms, we may still need to resolve to out-of-band- communication and manual configuration in some cases.

5.5 Validation

To validate the privacy enhancements, we created a small test environment of four users to simulate a group of privacy-conscious peers. The goal was to verify the effectiveness of the privacy enhancements by analyzing what could be deducted from the traffic. Furthermore, we hope to get a better understanding of application privacy in general, including factors beyond the control of the prototype that may impact the privacy of users. The prototype, with the privacy enhancements, was deployed on hand-held Nokia N810 Internet tablets. Each tablet was equipped with a unique identity, and the Internet call application configured to use it as the SIP account. The tablets were given to a set of test users, who were asked to use them for instant messaging, voice and video calls.

The tablets were connected using a standard IEEE 802.11b Wireless LAN (WLAN) access point with a Dynamic Host Configuration Protocol (DHCP) server to provide IP addresses within the 10.0.0.0/8 private domain. The prototype was configured to use HIP for session security. To simulate a P2P overlay, we used LAN broadcast as the back-end for the distributed storage and lookup. To evaluate the enhancements, we recorded and analyzed the network traffic generated by the tablets, simulating the worst-case scenario where an intermediate can log all overlay infrastructure traffic. The purpose was to analyze what can be deduced from the logs before and after the privacy enhancements.

Initially the prototype running on the tablets was set to the Open privacy mode. Although the keys used in the lookup are hashes of the SIP AORs (and not readable identifiers), the identifiers they correspond to can be read from the clear-text response. After a short while, we had compiled a mapping of these, together with the IP address used by each peer (presented in Table 5.1).

Table 5.1. The hash to SIP AOR relationships found, with responsible IP address.

Hash	SIP AOR	IP address
I6XlisZMhWcfO7gdVni4HdGZLbA=	alice@p2psip.hiit.fi	10.0.0.64
uDOI1fxZGRC4ghvHrbGSx+Ia6xM=	bob@p2psip.hiit.fi	10.0.0.68
vd4o2lZJ/yAVY9+pgU+Fz9Uh+PA=	carol@p2psip.hiit.fi	10.0.0.48
/+aYyc+gJMwwcgRoV3QoBcdyGfk=	dave@p2psip.hiit.fi	10.0.0.54

A sample from the recorded traffic is presented in Table 5.2. From the log we see Alice establishing a connection to Carol at 157 seconds. Even though the actual data traffic is secured by HIP, the pattern of relatively short IPSec ESP bursts provides clues of an instant messaging session. At 182 seconds, we see how Dave contacts Bob. The continuous flow of ESP suggests a voice or video call. At 195 seconds we see traffic again, most likely instant messages, exchanged between Alice and Carol.

After the initial session, the privacy mode was set to Paranoid, and the devices rebooted to reset any existing IPSec security associations. A short sample from the traffic is shown in Table 5.3. Although we cannot determine what was sought at 91 seconds, it is fairly certain that a call was made between Dave and Alice. The lookup response is encrypted, but the size fits within what we would expect of an encrypted registra-

Table 5.2. Samples of the traffic log before the privacy enhancements.

Time	Source	Target	Data
..			
157	10.0.0.64	(all)	Lookup(vd4o2lZJ/yAVY9+pgU+Fz9Uh+PA=)
157	10.0.0.48	10.0.0.64	Registration package for Carol (1527 bytes)
157	10.0.0.64	10.0.0.48	HIP BEX (HIT1 to HIT2)
162	10.0.0.64	10.0.0.48	Small burst of ESP
162	10.0.0.48	10.0.0.64	Small burst of ESP
165	10.0.0.64	10.0.0.48	Small burst of ESP
165	10.0.0.48	10.0.0.64	Small burst of ESP
..			
182	10.0.0.54	(all)	Lookup(uDOI1fxZGRC4ghvHrbGSx+Ia6xM=)
183	10.0.0.68	10.0.0.54	Registration package for Bob (1530 bytes)
183	10.0.0.54	10.0.0.68	HIP BEX (HIT3 to HIT4)
190	10.0.0.54	10.0.0.68	Continuous flow of ESP
190	10.0.0.68	10.0.0.54	Continuous flow of ESP
195	10.0.0.64	10.0.0.48	Burst of ESP
195	10.0.0.48	10.0.0.64	Burst of ESP

tion package, and the data traffic matches the pattern of a voice or video call. The IP addresses used for the lookup and response reveal the peers involved, assuming the hosts were provided the same addresses by the DHCP server as before. Furthermore, the HITs used in the data traffic reveal with certainty the identities of the communicating peers.

Table 5.3. Samples of the traffic log with the privacy enhancements in use.

Time	Source	Target	Data
..			
91	10.0.0.54	(all)	Lookup(J8doPupTwui0sugdTnC0DNkEggo=)
91	10.0.0.64	10.0.0.54	Encrypted data (2796 bytes)
92	10.0.0.54	10.0.0.64	HIP BEX (HIT3 to HIT1)
99	10.0.0.54	10.0.0.64	Continuous flow of ESP
99	10.0.0.64	10.0.0.54	Continuous flow of ESP

To simulate the use of additional source-address- hiding techniques, the HIP Host Identity database was reset on all devices, and the DHCP server configured to provide addresses from a different IP range. A sample of the traffic is shown in Table 5.4. We can still see that a voice or video call is made, but the peers remain unknown. We cannot identify the identifier for whom registration packets are sought, and the content of the response is encrypted. The IP addresses are also unidentifiable, as are the HITs used in the HIP BEX.

As the traffic analysis shows, the privacy enhancement does a fair job ensuring that the data managed in P2P communication systems reveal as little as possible on the application level. However, for complete privacy, we need to consider other factors as well. Using HIP to secure the data

Table 5.4. Samples of the traffic log with the IP and HIT reset, and privacy enhancements in use.

Time	Source	Target	Data
..			
122	10.0.0.18	(all)	Lookup(2qJJ8rIlbzgk5QivRZ8PfZ4XSB0=)
124	10.0.0.12	10.0.0.18	Encrypted data (2796 bytes)
124	10.0.0.18	10.0.0.12	HIP BEX (HIT5 to HIT6)
130	10.0.0.18	10.0.0.12	Continuous flow of ESP
131	10.0.0.12	10.0.0.18	Continuous flow of ESP

connections provides many benefits, but identifies the end-points to outsiders as well. Even without HIP and IP, hardware Media Access Control (MAC) addresses, or other host identification schemes might be used for the same purpose. Although the session data is encrypted, traffic analysis may reveal the type of content. However, these issues are considered out of scope for our work, as they relate to the general problem of communications privacy, and are being addressed by work such as the SlyFi [55] design. Our focus is only on the data managed by the distributed storage, and what it reveals; an issue specific to P2P systems.

5.5.1 Optimizing the data management

Although our model does well in protecting the privacy of users from intermediate nodes, it has tradeoffs. It increases the amount of data stored in the overlay, as the registration packets need to be published separately for each peer. Furthermore, it also requires that the peers perform a large amount of cryptography. Even though the packets contain only the information needed to connect, and are therefore relatively small (commonly a few KB of uncompressed data), the overhead is noticeable and can become overbearing.

As presented in Chapter 4.3, assuming an average of 190 contacts per user, periodically updating a unique, 3.2 KB, registration packet for each one would require transferring 608 KB per update, or over 6.5 MB per hour assuming a five minute refresh period (eleven updates per hour). Even though these loads may be manageable for high-end mobile devices, they still create an unnecessarily large burden on the storage system. Applying compression before encrypting the data may ease the storage requirements, but it does nothing to lessen the amount of requests the storage service has to process, as each entry still needs to be updated separately.

A simple, but efficient, optimization is to publish only *links* to the registration packet for each user, while the registration packet itself would only be published once. On initialization, a random index key for the registration packet would be generated (the packet key). Furthermore, a new, symmetric, encryption key would be generated, used to encrypt the packet before it is published under the packet key. For each user, a small *link* packet would be produced that contain the packet key, and the symmetric encryption key needed to decrypt the content. This link packet would be encrypted using the identity key of the contact, and published under an index key according to the privacy scheme.

When retrieving the registration packet for a contact, only the link package is retrieved initially. The peer decrypts this using his identity key, and finds the packet key and the encryption key used for it. The space saved using this method is substantial: assuming 1024 bit RSA identity keys, and sizes for the packet- and encryption keys so that the link package can be encrypted using the identity key directly (e.g., 512 bit encryption key and a 160 bit packet key), the link package would, theoretically, occupy only 128 bytes. For the average user with 190 contacts, it means that only 24.3 KB of storage is needed to store the registration packet for all contacts using the privacy scheme. Compared to the unoptimized approach, this represents a drop of 96.0%.

Although the space savings are substantial, more importantly is the fact that updating the registration packet for all contacts requires only a single request. Neither the encryption or package key, contained in the link packages, need to be updated, only the registration packet they point to. Therefore, after the initial setup (creation and publication of the links), the link- scheme does not, in theory, generate more traffic then when not using the privacy enhancements at all. The cryptographic load on the hosts decrease as well, as there is only one packet to encrypt and sign for each update. In practice however, the link packages need to be periodically refreshed, depending on the type of storage service used.

5.5.2 Data usage

In order to confirm our assumptions regarding the data use, we recorded the traffic related to the publication of the registration packets when using the different privacy modes. The set-up was similar to what is described in Chapter 4.3 to quantify the load on the storage service. We had a single peer, configured with 190 contacts, using a web-based storage service. This storage service application was configured to record the amount

Table 5.5. Results of the storage use measurements using the privacy enhancements.

	Disabled	**Normal**	**Linked**
Requests, total	133	25269	2416
Requests, per hour	11.1	2105.8	201.3
KB transferred, total	422.5	81972.6	2560.1
KB transferred, per hour	35.2	6831.1	213.3
Stored entries	1	190	191
Stored data, KB	3.2	616.4	189.3

and size of each request the client made, as well as the total number of entries stored in its database.

The data traffic was recorded over a 12 hour period using three different client modes: with the privacy extensions disabled, with the privacy extensions enabled using per-contact updates, and with the privacy extensions enabled using the link- scheme. The registration was published with a short validity and updated once every five minutes. The link packages (when using the link- scheme), however, were published with a long expiration time and updated only once an hour. The results of these measurements are presented in Table 5.5.

As the results show, the values measured as in line with our assumptions. The differences between the expected and measured values in the optimized mode is due to the data formats used in the prototype. Each of the link- packages used the prototype's generic XML package format, which increased the size of the entries from the theoretical minimum. However, as the results show, even without further compression, the data usage of the optimized scheme does remain within reasonable limits. With an average of 201.3 requests per hour (or 3.4 requests per minute) and 213.3 KB of data transfer per hour, it is manageable even for resource-limited mobile devices.

5.6 Summary

This Chapter has dealt with the privacy issues in P2P systems based on persistent, strong, identities. We have highlighted the problems and presented a simple mechanism for hiding our activities, while adhering to a very generic way of accessing overlays. By not requiring changes in the structure or logic of the overlays, this mechanism can be applied to most existing systems.

The value lies in acknowledging the problem, and tackling the usability issues our solution (or similar ones) unavoidably leads to, due to the different operating modes (which are further examined in Chapter 8).

The model we have presented can be used as a blueprint for new, or existing, systems based on similar philosophies (such as P2PSIP). How it is implemented can vary depending on the nature of the application, but our model outlines the problems, and possibilities, there are. Our model shares many of the aspects of current technologies for privacy enhancement (such as Freenet[32], TOR [41] and OneSwarm [75]) in that it provides protection from eavesdroppers, whether these are within the overlay or outside of it. It does not provide attribution- free publishing and consumption of content, although this could be achieved by applying one of the technologies previously listed to the overlay. The main advantage of our model is that it does not require changes to the overlay, and is, to the best of our knowledge, the first attempt at providing a complete solution, not only to the technical challenges, but also usability issues.

6. Secure P2P Web

The World Wide Web (WWW) is arguably one of the most popular services on the Internet. The first *web pages* were simple: a bit of textual information with embedded hyperlinks coupled with small graphics, authored by a single author and delivered by a single server. As explained in Chapter 3.3.2, these are well suited for P2P delivery, similarly to other services based on HTTP. However, modern web pages are more complex and the user's experience of "the web" is often developed from myriad components from a variety of providers and systems. For instance, a simple blog post might include (*i*) content from the blogger, including the posts themselves, a set of thematic images and backgrounds, etc., (*ii*) content from the blog hosting services, which could include navigational aids, logos, etc., (*iii*) content from third-parties associated with the blogger or the hosting service (e.g., advertisements) and (*iv*) content from numerous readers who left comments to the given post. The number of actors contributing to a conceptually simple "web page" is potentially enormous.

Security in the WWW architecture is based on authenticating, and trusting, the source server and securing the data during transport. The traditional assumption is that the content as secure as the server hosting it. Within the context of P2P web, it means that we must grant the same trust to each peer hosting any piece of content we access, or alternatively, only access content from those we trust. This undermines the value of P2P, especially in open systems, as we might not even know most of the peers participating in the network. We need a way of securing the content of the P2P web, so that not only static web resource, but also interactive web applications can partly be hosted on untrusted peers. Secondly, we need to address the availability of web resources, so we do not have to rely on only a few trusted peers to access the content.

In this chapter, we introduce a model to secure the P2P web by adding integrity and accountability to not only web pages, but also individual elements. This not only allows web resources, and applications, to be hosted by untrusted peers, but also protects the user from possibly malicious content on these pages. Furthermore, we present a novel scheme for cooperative P2P Web caching. Together these construct a secure and accessible P2P web environment.

6.1 Securing Web content

The approach we have taken to secure the P2P Web is by using the strong identities provided by the framework, and utilizing the flexibility of HTML's syntax to implement integrity and accountability into web pages. This allows for an unobtrusive, backwards- compatible way of verifying web content.

6.1.1 Page structure

The first aspect of securing content concerns protecting the integrity of the overall web page, which in our framework is done by using the author's private key to cryptographically sign the markup document. This signature is included in a new HTTP header, along with the author's public key (the identity). Protecting the integrity of the page structure is not crucial for the securing the individual content components (discussed below), but only for conveying the overall intended composition of those components.

Providing over-arching page integrity is similar to signing data packets published in the overlay, except that the integrity (or lack of) is visible to the application (web browser). This allows for flexibility, as we can verify the content even if it is not distributed by the original author. Furthermore, the application can, in some cases, choose to ignore it in case the content components are intact, as we will shortly see.

6.1.2 Content components

The HTML syntax is based on *tags*, such as `<div>`, that both structure the document and define part of the appearance of elements. These tags support attributes containing information that can be used for anything from indexing the page, to defining behavior. Some attributes, such as `id`, are standardized, but in practice HTML allows liberal use of custom attributes which have application-specific meaning.

The basis of our approach to securing individual content components is to add cryptographic signatures to content blocks within the web page. Different blocks can therefore be authored by different users with integrity and accountability information that directly map the content to the author. This gives browsers and end users the ability to enforce policy decisions on what components to render, omitting possibly malicious components based on their knowledge of the authors. The signatures are included in custom attributes of the HTML tags, making them rendable by nearly all legacy browsers (which will ignore unknown attributes).

We offer two approaches for securing these elements. First, the signature can be interpreted literally, meaning that the signature is made from the actual markup within the tag. This ensures that the content of the tag is precisely what the author intended and allows for no possibility of the site maintainer customizing the look or formatting of the content. This tradeoff may well be useful in some circumstances. The second approach is to add the signed content as a custom attribute to the tag, and use it for *decorating* the content when rendering the page. The data is provided in a safely encoded (base64) format, which is decoded, validated and inserted into marked locations within the content.

Although technologies such as Cascading Style Sheets (CSS) provide a degree of customization without changing the markup of a page, the second approach gives the maintainer more freedom in designing the site's look and feel because it separates the content from the formatting. However, in some cases the formatting of the content is fundamental to the content (justifying the first approach). For instance, consider an HTML table. In this case, the content author may wish to secure the formatting to ensure that rows and columns are constructed properly. Our framework treats both options as valid and leaves the decision of which to use to the author.

In addition to signing the content in the markup, we may include signatures for external objects. For instance, we may include images, scripts or other content from outside the system (the normal "centralized" web).

6.1.3 User-generated content

In order for the the content securing scheme to be useful, we need a method for users to produce signed content suitable for it. This we achieve by cryptographically signing user-submitted data to a web page (such as a blog submission) with the user's identity. This enables the web site to receive signed blocks of data, which it can then be used as a secure content component accountable to the author. This does require careful planning of the web site structure so that the submitted data will be usable as content. This is achieved by defining the format of the custom data tags to match the HTTP POST syntax.

6.1.4 Security policies

So far, we have only discussed the mechanics of methods for verifying the source of web content, without considering how that information is used or affects the web browsing experience. As explained in Chapter 3.2.1, the framework provides information about the identities, from which we can asses the trustworthiness of an actor. Content from trusted sources is fairly easy to handle. However, there are different approaches to dealing with untrusted content. The harshest option is to block untrusted content completely, not rendering or loading it at all. Although effective against potential threats, this easily cripples the browsing experience, rendering the web useless. A less sever option is to *sanitize* the content by turning all or part of the markup into plain text. As for the presentation, we could prevent certain elements which can result in disturbing the layout of the site (such as positioned `<div>`s), while allowing changes in text color or other minor modifications.

Functional components—such as embedded JavaScript or external Flash applications—could be either disabled completely or allowed to run in a restricted environment. Sandboxing such content is effective but requires close cooperation from the run-time environment. JavaScript, due to its dynamic nature, allows partial restrictions to be applied within the browser. We could for instance disable popups, network access ("Ajax") or browser redirects. Obviously, when content is received with an invalid signature great precaution should be taken, preferably not rendered at all. The particulars of how which of these restrictions are imposed and how they are applied are *policy decisions* and should be set according to the browser preferences (either set by the user or an administrator). We stress that these policies are *local* and not assigned by remote web pages or components thereof.

6.1.5 Implementation

As part of our research, we created a prototype implementation to gain a better understanding of the feasibility, technical challenges and usability of our model. The prototype was implemented as a Mozilla Firefox plugin for Linux, which connects to the P2P framework through the interfaces described earlier, in order to access the identity and relationship database. The plugin consists of three parts; an XPCOM[1]component that processes the raw HTTP streams, a JavaScript application that alters the rendering and a small user interface for displaying security information and for controlling the security policies of the prototype.

As a Firefox plugin, the development and deployment was considerably easier than actually modifying the browser, but this model did also impose restrictions which meant that all features could not be completed. The prototype has, however, provided a initial understanding of how the model we sketched could be implemented, and the technical challenges.

Page signing

We use custom HTTP headers for both indicating support for the mechanism and for carrying page signatures. Support for the scheme is indicated (both by the client and server) with the *X-OP-Supports: true* HTTP header. Page signatures are inserted in the *X-OP-Signature* HTTP header (SHA-1-based RSA signatures in our prototype). Finally, the *X-OP-Key* HTTP header contains the user's public key, base64 encoded.

The stream processor inserts these headers in every request made by the browser, signaling to supporting servers that pages should be signed. The browser plugin, acting as a Firefox *stream decoder*, verifies the page signatures before passing the content to the renderer. The result of this verification is stored in the instance variables of the page window, accessible for the other components of the plugin.

The stream processor also captures and signs the data of HTTP POSTs made by the browser to supporting sites (enabling user-generated content to be secured). These signatures are carried in the same headers as used by the server for the page signatures.

Content processing

Our plugin alters the rendering process by executing a JavaScript appli-

[1]XPCOM (Cross Platform Component Object Model) is the component model used by Firefox to expose much of the functionality (as object components) of the browser to plugins, and allow custom components to be added or replace existing ones.

cation after constructing the initial DOM tree. This approach has flaws (allowing possible malicious JavaScript or content to be loaded), but allows post-processing of the page, and alterations similar to what a proper implementation would do.

Our prototype supports only the second of the two methods of securing content elements discussed in Section 6.1.2—by including both the data and signatures in element attributes. Three attributes are used: *op_key* contains the public, *op_signature* contains the signature, and *op_data* for the actual signed data. Both the signature and key are (as in the HTTP headers) base64 encoded. The data attribute contains the data as URL-encoded key-value pairs as this is the format in which we encode HTTP POSTs. Therefore, data posted by users can be used unmodified in these blocks.

These tags were rendered by *decorating* the content instead of completely replacing it. After verifying the signature, the plugin uses the key-value data to complete fields within the tag. The target locations are found by matching the keys of the source data to the element identifiers (*ids*). As these are page-unique, we use the parent tag ID as a prefix to create an unique namespace within it. For instance, a <div> with an ID of *msg01* and a data key *title* caused the renderer to replace the content of the child element with the id *msg01_title*. To support better customization of the appearance, the rendering processor adds the result of the verification as an attribute *op_status* to the element. This is used to select a suitable style when rendering. Figures 6.1 and 6.2 illustrate this process. Figure 6.1 displays (truncated for readability) a signed <div>, which is rendered (when judged as *trusted*) as shown in Figure 6.2.

```
<div id="sdiv5" class="entry"
     op_data="header=Hi&message=Testing+123"
     op_signature="OyjONQTCAR6Mv/sBjRaF.."
     op_key="LS0tLS1CRUdJTiBQVUJMSUMgS0..">
   <div>Posted 11:43:51</div>
   <div id="sdiv5_header"></div>
   <div id="sdiv5_message"></div>
</div>
```

Figure 6.1. The HTML source of a signed block.

```
<div id="sdiv5" class="entry"
     op_status="trusted">
   <div>Posted 11:43:51</div>
   <div id="sdiv5_header">Hi</div>
   <div id="sdiv5_message">Testing 123</div>
</div>
```

Figure 6.2. The signed block after processing.

```
div.entry[op_status="trusted"] {
     background: green; font-size: large;
} div.entry[op_status="invalid"] {
     display: none;
} div.entry[op_status="untrusted"] {
     background: red; font-size: small;
}
```

Figure 6.3. A CSS style sheet declaration highlighting the trustworthiness of the content.

As only the content of certain elements is modified, it allows the site maintainer to control the visual appearance. Using the *op_status* attribute, the site maintainer can provide a style sheet (such as the one illustrated in Figure 6.3) highlighting in a site-specific manner the trustworthiness of the content blocks.

Before rendering the data values, the plugin *sanitizes* the data according to security policies. In our current prototype, we support only full sanitation (escaping all markup) which normally is applied to all content, although the user can choose to bypass this from trusted personas (i.e., that have a good trust score).

Control interface

The control interface of our prototype is used to display information about the current page and the signed tags, as well as to control how the content is rendered. The interface is implemented as a small popup-menu located at the bottom status panel, showing the trust status of the current page, similar to HTTPS indicators. The controls allows the user to block or completely hide untrusted and invalid data blocks, and choose whether to sanitize content from trusted identities.

The prototype uses the relationship information got from the P2P framework to classify the page as being either *trusted*, *untrusted* or *invalid*. In addition to this classification, the interface can display a simple human-readable description of the track record, such as *You trust this person, knowing him well (through browsing).*

External objects

Our approach to external objects (such as media files or JavaScript source code) embedded in a page mirrors our handling of the tags, by adding *op_signature* and *op_key* attributes to elements such as `<img>`, `<script>` and `<object>`.

Due to how the page rendering in Firefox is structured, we are not able to intercept or prevent these external objects from being loaded. However, we experimented with post-processing images, changing how they are displayed based on the trustworthiness of the keys. As with content tags, images can be hidden completely, blocked or displayed with a warning.

Server library

As our mechanism only affects the HTTP headers and HTML content, it can be implemented on servers using server-side scripting (without changes to the HTTP server itself).

We created a PHP library for Apache's HTTP server which automates the page signing process. Applications need only initialize the library with their private keys, and call *flush()* at the end of each page transmission. The library connects to the P2P framework, and uses its identity to sign the content.

6.1.6 Related work

Our mechanism is similar to the scheme used by different browser plugins (such WebPG [4], Mailvelope [1]) to sign or encrypt either whole pages or individual sections. These typically focus on web mail, but some support more generic use. However, the focus of any of these is only on validating the source and data integrity, without considering how the content is handled. The schemes appear also too rigid to be easily adapted by web applications in a systematical manner.

Sandboxing individual HTML elements is discussed in [2] and shares ideas with our approach. In particular, the idea of downgrading the privileges of individual content components. However, [2] considers only how elements could be protected from each other and does not consider protecting the user from the content.

To our knowledge, our scheme presents the first systematical approach for increasing security by adding accountability to the web that is compatible with modern sites featuring user generated content. With this approach, we area able to decouple the web page's trustworthiness from that of the serving node's, making it possible to securely host content anywhere. This is important for P2P web, as replication is crucial for accessibility and performance due to the relative instability of peers compared to traditional Internet servers.

6.2 P2P Web cache

Having addressed security concerns regarding web content in an environment consisting of untrusted peers, we focused on addressing availability in P2P web. We approached the topic through the use of distributed web caching, instead of active content distribution, as it provides a simpler, unobtrusive, way of replicating content. Peers do not need to retrieve and host content they have no use for themselves, which lowers the threshold for participating in the absence of a working incentives mechanism.

As reviewed in Chapter 2.8.1, there exists multiple solutions for building a distributed web cache, and building one for P2P networks is only a matter of engineering. However, using a traditional approach to distributed caching results in a system with a large amount of maintenance traffic. This may not be a problem for the intended audience of the proposals presented in Chapter 2.8.1 (such as data centers), but are problematic for devices such as mobile, battery- operated, handsets. Therefore, we decided to focus on the data management in order to identify what the parameters are that affect how data should be cached, in order to optimize the maintenance of a distributed cache in P2P networks.

6.2.1 Data management strategies

The initial P2P web caching scheme we set out with follows a very straightforward approach to distributed caching. Each piece of content cached results in a single entry in the overlay (advertising the availability of it). The size of the local content store is managed using a simple least-recently used (LRU) algorithm. As the storage fills up, the content that has been least recently accessed will be discarded. Although this elementary data management scheme does demonstrate the usefulness of distributed caching, it has its drawbacks.

The diversity of the available content suffers. As the content within the cache is replaced using a local LRU, it leads to popular sites being over-represented. As a large number of the users will cache the same, frequently accessed content, it will *push* out less popular items. A better approach would be to use a cooperative replacement algorithm, which considers also the popularity of the content within the whole P2P community. For instance, a specific percentage of the LRU stack would be dedicated to *rare* content. When expelling content from the cache, these would be spared even though they have been less used than other content.

Table 6.1. Dissection of the expire-times indicated in the HTTP headers of cacheable content.

Expire	Objects	% of cacheable content
0 seconds	365	13.8%
5 min or less	5	0.2%
2 hrs or less	106	4.0%
24 hrs or less	69	2.6%
1 year or less	315	11.9%
76 years or less	1189	45.0%
> 76 years	596	22.6%

Furthermore, as each one of the cached content items (each URL) produces an entry in the overlay storage, the scheme generates a large amount of entries, and traffic. A single web page can consist of dozens of images, style sheets and other components that all need to stored and individually advertised in the overlay. This could be improved using different types of content *grouping* based on the URLs and context. In order to optimize the data management strategies for the proxy, we recorded and analyzed the traffic log of a common[2]web-browsing session. Over the course of 30 minutes of active web browsing, 3771 HTTP requests were made, and over 34 mega bytes (MB) of content data received. Of these requests, 2643 (70%) were considered cacheable (according to the HTTP headers), which accounted for over 25 MB (73%) of the data. Furthermore, of these cacheable objects 80% were marked suitable for long-term (over 24 hours) caching (as presented in Table 6.1).

Although these figures represent only a single, specific, web-browsing session, they show that, at least for the sites visited, there is a large amount of content that can be cached for a longer period. This indicates that a cooperative caching scheme is not dependent on instantaneous, or quick, propagation of updates to the cache index in order to be useful. As much of the content can be cached for more than a day, we can delay the

[2]We recorded the requests and meta data of a 30 minute long web browsing session consisting of visits, and engagement, with popular sites from the Alexia top 500 sites (http://www.alexia.com/topsites). Although this obviously provides merely a snapshot of how a specific user used a particular set of sites, the log is used to provide clues into what the parameters are that can be optimized in a cooperative caching scheme.

individual advertisements, and perform batch-updates instead to minimize storage- and traffic overhead. Analyzing the request log further, we see that it consists of irregular bursts instead of a constant stream of requests (Figure 6.4). This is, of course, expected as requests are made only when the user loads a new page, after which there is a quiet period when the page is viewed.

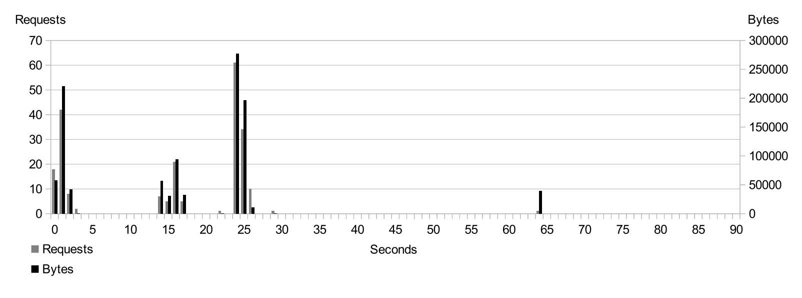


Figure 6.4. A snapshot of the traffic pattern of a typical web browsing session. The requests appear in bursts, as the user loads new pages. Each new page generates multiple requests as images, scripts, and other linked content is loaded.

Inspecting our traffic log, we find that the bursts found there consist on average of 37.2 request to 1.7 different hosts over a 2.8 second period. Although this varies depending on the site structure, the ones present in this log seem to be using only few hosts for serving each page. A simple improvement would therefore be to group the cached content according to the URLs. For instance, all content within the same URL path, or from the same host, could be grouped together in a single advertisement. The overlay key for this would be constructed of the URL path or host name. Although the scheme would decrease the number of entries in the overlay, it would also lead to more frequent updates to those, as the addition or removal of a single item within the group requires a new advertisement. However, as indicated by Table 6.1, much of the content have so long expiration times that we can afford to delay the advertisements without rendering the scheme useless.

Inspecting the logs further, we find that the average pause between these bursts is 152 seconds, slightly over 2.5 minutes, meaning that on average, the user spent 2.5 minutes on each page. In order to efficiently group advertisements according to host (or site), we need to be able to predict when the user has is unlikely to generate more requests to the site, as that would be the optimal time to publish the advertisement. By analyzing the pauses between accesses to content from the same site, we find that the vast majority (96%) is accessed within ten seconds of the previous request (due to the request bursts). However, of the remaining, 78% is accessed within 220 seconds, as shown in Figure 6.5.

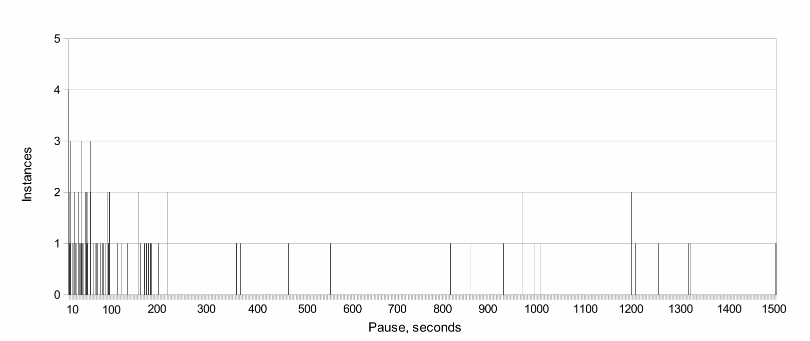


Figure 6.5. The number of pauses between 10 and 1500 seconds between requests to the same host. For instance, a 1200 second pause between requests to a single host were encountered two times.

This shows that for the browsing session we are inspecting, after 220 seconds had passed since the last request to a particular host, it was highly unlikely that the user would visit that site again. In order to more precisely measure the potential benefits from grouping, and delaying the cache advertisements, we created a simulator for the caching mechanism. The simulator performed a chronological walk-through of all requests, grouping the cacheable objects based on the source host. After a specific time had passed, during which there had been no new entries from the host, the advertisement was *published*, with the counters re-set. Figure 6.6 shows a graph over how the total number of cache advertisements, and the average accessibility delay, were affected when varying the publishing delay. The accessibility delay is the time between when an object has been received, and when a advertisement containing it is published.

As the figure shows, the overhead (amount of advertisements) drops rapidly when the delay is increased up to 100 seconds, after which it slows down and eventually settles at around 135. With 110 unique hosts in the sample, this means only a 23% overhead compared to the ideal (one advertisement per host), and a 51% drop from the worst-case scenario (without any delay). However, as the accessibility delay is directly tied to the publishing delay (as it is always equal or greater), it does rise linearly, prompting us to select the publishing delay based on a good balance between the overhead and accessibility.

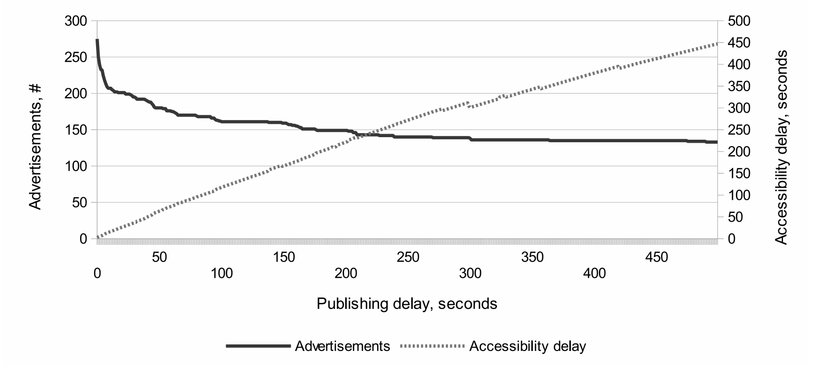


Figure 6.6. The total number of cache advertisements published, and the average accessibility delay depending on the length of the publishing delay. Total number of unique hosts in this sample was 110.

6.2.2 Self-adjusting expiration

In order to automatize this process, we created a self-adjusting caching algorithm (presented in the Appendix) that strives to optimize the accessibility without increasing the overhead. It is initialized by providing two values describing the accessibility that the user is striving for: the *maximum* delay and the *average*. The maximum delay (Cache.MAX_DELAY in the listing) is the ceiling for how long a cached object is kept without being advertised, while the average (Cache.AVERAGE_DELAY) is the average time the user is striving for. The algorithm keeps track of the publishing times, and adjusts the publishing delay accordingly in order to converge the average towards the user-defined value. Furthermore, it records the time at which each object is added, and forces a advertisement to be published after the user-defined maximum amount of time has passed, unless the objects have been already advertised due to a publishing delay- timeout.

Furthermore, it tries to optimize the accessibility by slowly decreasing the publishing delay, even though the accessibility delay is less than the user-defined value. But only until it encounters an advertisement “miss”; an object that would have been included in a previous advertisement, had the publishing delay been greater than the current value, but less than the user-defined average limit. With this addition, the algorithm will decrease the accessibility delay, but only until the overhead (the number of advertisements) increases, at which point it will *back off* by restoring the publishing delay.

6.2.3 Validation

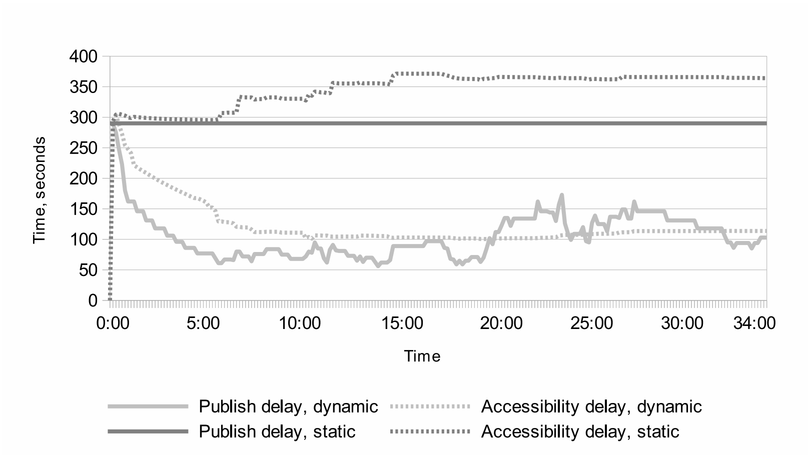
In order to measure the performance of this algorithm, we recorded over 34 hours more of active web browsing over the duration of one week, which we analyzed using a Python implementation of the algorithm. The web browsing logs were obtained by using a HTTP proxy that recorded the time of each request, complete with the HTTP headers relevant to caching. From these entries the un-cacheable requests were removed, and the remaining used as data for the simulator implementation. The logs were collected from two separate hosts during daily web usage and combined into a single continuous log (with inactive periods removed). As before, although this analysis reveals only how well the algorithm performs against the web browsing patterns of the test subjects, it does provide a view into what the tradeoffs are when optimizing the accessibility.

Figure 6.7(a) shows how the accessibility delay is varies during the course of the sessions, and how the publishing delay is adjusted accordingly when the target accessibility delay is set to 290 seconds. The graph shows also how the accessibility delay would develop without adjustments (the *static* mode), when the publishing delay is fixed at the 290 seconds. Figure 6.7(b), on the other hand, shows the cumulative amount of advertisements produced by the two modes, as well as the ratio between the self-adjusting (*dynamic*) and static mode.

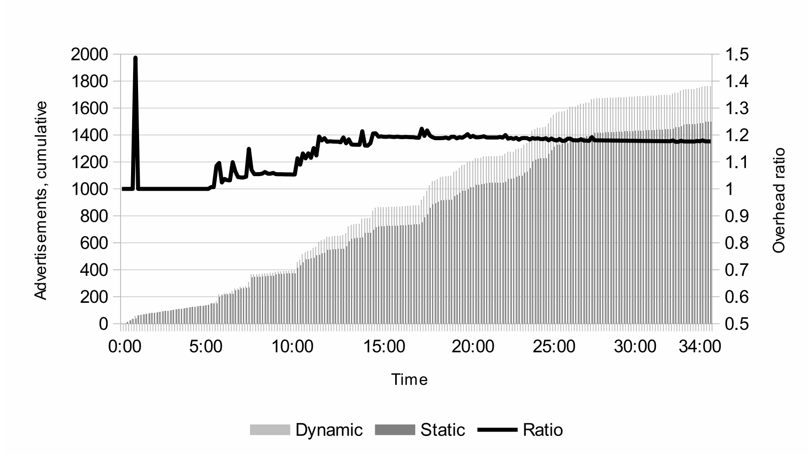
Over the course of the sessions, the self-adjusting (*dynamic*) algorithm produced 1762 advertisements, with an average accessibility delay of 113.8 seconds, while without adjustments (publishing delay set to 290 seconds) it produced 1498 advertisements, with an average accessibility delay of 364.1 seconds. For these specific sessions, it means that the self-adjusting algorithm increased the number of advertisements by 17.6%, while decreasing the accessibility delay by 68.8%. Overall, we can conclude that at least for the types of browsing behavior we analyzed, the self-adjusting algorithm decreases the accessibility delay substantially while moderately increasing the overhead.

6.2.4 Security considerations

The P2P web cache provides one approach to increase content availability and reduce stress on centralized nodes, but includes also a number of serious security threats. An obvious threat is that the content advertised cannot be verified. Without any protective mechanisms, malicious users



(a) The development of the accessibility delay.



(b) The cumulative amount of advertisements.

Figure 6.7. The development of the accessibility delay and publishing delay, as well as the cumulative amount of advertisements produced, during the web browsing sessions when using the self-adjusting algorithm. The values for the static mode is provided for comparison.

are free to advertise the availability of popular content which could in fact be viruses or false data. Through the security scheme presented in Chapter 6.1, we can verify that the content is what the author intended, and limit the damage of untrusted content through sandboxing and sanitation. But this does not prevent peers from wasting time and bandwidth retrieving falsely labeled data.

Currently our prototype fetches content only from peers that have been verified by a trusted identity authority. Although this limits who is able to claim having possession of any content, it relies on the integrity of the identity authorities, which can vary. A more personal solution is presented in Chapter 7, originally intended to prevent unwanted traffic, that can also be used for deciding which peers to use as content sources in P2P web caching.

In addition to the threat of distributing false content, P2P web cache raises privacy concerns. As the proxy publicly advertises the content cached by the user, it acts as an open browsing history log. Furthermore, as some sites may adapt its content or content URLs to a specific user, it may contain sensitive data unless the HTTP headers have been appropriately set. But the issue does not only affect the user caching content. The users requesting the content expose as well their browsing habits, although to a lesser degree. This touches on the general issue of web anonymity, which is hard to solve, especially as our framework is based on relationships and strong identities.

It is clear that we should be very careful when choosing what data to cache. This includes scrutinizing the content and related metadata (such as HTTP headers), in order to find clues to the nature of the content. Furthermore, it could also involve changing the browser's behavior. Similarly to the anonymized (or *incognito*) modes offered by many web browsers, we could have an *open*, or *sharing*, mode with clear visual clues. The browser would warn the user (or switch automatically) whenever browsing sites not meant to be cached (for instance, sites with a login form).

6.2.5 Related work

As reviewed in Chapter 2.8.1, there exists several solutions for decentralized, or P2P, web caching. These are designed as either more scalable alternatives to traditional, centralized web caches, or for environments without a dedicated infrastructure. They rely on close cooperation between the nodes, and concentrate on efficient distribution of data (commonly using a set DHT algorithm). Our focus is not on the structure of the network or efficient data distribution, but on optimizing how data availability is announced in networks without a set structure. This can be coupled with other distributed web caching solutions in order to minimize the maintenance traffic.

6.3 Discussion

For P2P web to be viable, we need to address the inherit security threats and availability issues. The web has become the most popular application on the Internet, and subsequently an important channel for malware distribution and other malicious activity. Currently we rely on the reputation of the site, and the organization behind it. However, this does not fully work even today, as we have seen cases where reputable sites have, inadvertently, hosted malicious content. In P2P web even this becomes void, as we, in order to increase availability, may have to rely on complete strangers to host content.

The framework we present has the potential to offer the security crucial for P2P web, but there are also downsides. These include the additional processing required to sign and verify web content, increases the size of web pages and the resulting increased load on peers. Also, the scheme may not work well with certain types of content, such as streaming media.

On the other hand, the mechanism works well with existing protocols, making it possible for servers to simultaneously serve clients that support the scheme and those that do not. Using the indicators in the HTTP request headers, we can select which peer receive the secured version, while omitting the signed version—and resulting overhead—for those that will not use it. Even if legacy clients do get the secure content, the syntax is backwards-compatible and users will have to rely on existing solutions (such as script blocking) for security.

To address the availability issues, we propose the use of distributed web caching, for which there are exists multiple solutions. Our contribution is to design a scheme which makes these safe to use even in untrusted environments, and to outline a strategy for optimizing the maintenance traffic. This is important especially for battery- powered devices, which is the focus of our P2P framework.

7. Preventing unwanted traffic

In the past years, unwanted and abusive messaging has penetrated almost all areas of the Internet, plaguing users by not only being distracting, but also by potentially carrying malicious content (viruses) and using up computing- and network resources. As we design a new networking model, we should address this problem, especially as the aim is to create a *secure* framework for applications hosted by the users themselves. These are often more vulnerable than traditional infrastructure, as they do not have the same resources to defend themselves. Many security threats in our environment have been solved by using secure data protocols (such as HIP) and a strong identity scheme (as described in Chapter 3.1.1). However, these do little to shield the user from unsolicited communication, *spam*, that would waste the already limited resources of mobile devices.

There is a clear incentive for spam. Perhaps the most widely studied manifestation is email spam, which has grown exponentially from virtually zero to a vast majority of all email traffic [117][163]. It is estimated to cause tens of billions of Euros in losses due to lost productivity, software crashes and identity theft [164]. Although counter measures, such as spam filters and virus scanners, have been deployed, some of the spam will always reach its goal. Even though this may only be a small fraction (as low as 0.00001% [86]), it is enough for the activity to continue to be profitable.

In communication systems, spam is not confined to merely emails or comments on blogs posts, but includes unwanted VoIP (Spam over Internet Telephony, or SPIT) and other unsolicited application data possibly infecting the target with malicious code. Especially mobile users would suffer due to the additional mobile data transfer charges and battery drainage. This problem has been anticipated, even dubbing VoIP as the next medium for spammers [35]. A number of products have been developed to address the issue (such as [45][107]), and members of the IETF SIP working group [73] have also recognized the problem, expressing their thoughts and possible solutions (as described in Chapter 2.4.1). However, these mostly rely on the presence of a centralized authority and are, in many parts, irrelevant for P2P environments.

Without a central authority, it is hard to keep track and block peers that engage in this activity. Detecting a rogue peer as it establishes VoIP calls to a thousand users simultaneously is easy in centralized environments. But it is impossible for a single user to notice this, as it has no overview of the whole system. The users of P2P networks can only rely on what they know themselves or have learned from others.

For our framework, we have studied the use of relationships for preventing unwanted and abusive traffic. This is well suited for P2P environments as it does not require a complete view of the whole system or a large, up-to-date, database of the tell-tale signs of malicious activity. We only need to keep a record of the peers we interact with, and exchange this information with others. Compared to existing relationship-based solutions, we focus on the privacy of users, and develop a probabilistic method for discovering social connections.

7.1 Relationship-based filtering

Relationships create a network of trust between users. Considering someone a *friend* implies that we trust that person to a certain degree. Filtering traffic based on these friendships, for instance allowing friends to connect unquestioned, while subjecting others to closer inspection, seems natural and straightforward. However, it leads to a very strict model where establishing new relationships is hard. We would need to use out-of-band channels and manual configuration to establish new relationships, or subject each peer to limited *trial* periods during which we monitor and assess the trustworthiness of that peer.

We have explored different ways of sharing relationship information for establishing trust between users, in other words, solving the traditional introductory problem in distributed systems. For instance, if Alice wants to connect to Bob, who does not know Alice beforehand (and is wary of accepting new connections), Alice might not get through. But if they both have a mutual friend Carol, Alice could use her as reference. By leveraging Bob's trust in this mutual friend, Alice would seem more trustworthy (or at least more accountable) and Bob would be more likely to accept the connection.

However, we wanted to extend the model from only one-hop *social links* (friend of friend, FoF) to multiple. We wanted a model where we could measure the distance to peers within our social network, and make decisions based on that information. In distributed systems, only the users themselves know who their friends are. Therefore, we need a way of sharing this information between users, so that the social links can be unveiled. This, however, raises two fundamental questions:

i How do we know whether the information is correct?

We need to verify that a claim of friendship is not made only by the subject itself, but also by the partner it claims to be friends with.

ii What is the incentive for participating, as it requires us to reveal our contacts?

We need to protect the privacy of the users. Few are willing to openly publish a list of all of their acquaintances. We need a way to anonymize this information while still be able to fairly confidently establish the path between users in a social network. This lowers the threshold for participating.

We approached the problem using a distributed relationship-sharing mechanism based on bloom filters.

7.2 Related work

Finding common friends is a fundamental feature of most social networking services. Furthermore, there has also been a number of attempts at creating independent systems for providing this service in a more generic manner for any application that may benefit from that information. The Pathfinder, designed by Juho Heikkilä, is a good example of such a system. Architecturally it is similar to the identity authorities described in Chapter 3.1.1. That is, a trusted third party service which can be distributed to several instances as long as both parties of a session trust the same instance. Therefore, it solves the two fundamental questions by having trusted Pathfinder nodes (information correctness), and by only using anonymized data (lowering threshold for participating).

The Pathfinder provides an interface for users to submit their contact list in an anonymized format. The anonymization is done by revealing only the digest value (hash) of the names of the friends. The Pathfinder can use these to establish a network between users (as the hash of a user name is consistent), even though it does not know the actual identity of those users. Connection initiating peers query the Pathfinder to establish a link between themselves and the target peer. The Pathfinder uses an efficient algorithm for traversing its internal database, and returns a signed statement containing the shortest path found. This is presented by the initiating peer during connection establishment[66][100].

Although the Pathfinder (or similar systems based on trusted nodes) provides a solution for discovering social links, it has a number of drawbacks. The most apparent being accessibility. Our framework is designed for distributed environments, where access to external networks or specific nodes may not always be possible. Secondly, it requires that both users as well as all intermediate acquaintances trust, and use, the same Pathfinder. The Pathfinder could to some extent be used in distributed systems, as it does not need to be a single centralized component. In overlay networks, we could share the responsibility amongst a number of trustworthy peers. These Pathfinders could also collaborate, creating an overlay between themselves for discovering social links. However, this does not solve the problem completely.

On the other hand, attempts at solving this problem by using fully distributed mechanisms often have a number of security concerns. For instance, the Loaf system uses viral propagation of Bloom filters constructed from contact lists[31]. Users create these filters using the email addresses of known, trusted, contacts, and attach them to the email they send. These are then used by the recipients for whitelisting email received from unknown addresses.

The loaf system has many of the traits we are looking for. It is fully distributed, and provides some degree of privacy. However, on closer inspection, the security is superficial. As the filters are publicly distributed, they are easy to obtain. Engineering a value to fit into a pre-defined Bloom filter is easy with any modern computing device. Even a low false positive probability of 1% would only take 100 attempts to overcome. Furthermore, it is possible to check for well-known addresses in the filters, revealing parts of the contact list. Furthermore, the loaf scheme extends only to friends of friends, which is less than we require.

7.3 BloomBuddies

The BloomBuddies scheme was designed to address these shortcomings by using a multi-level viral model to spread relationship information and a novel procedure for discovering common friends using Bloom filters. The viral distribution mechanism is simple; when establishing a peer connection, both users share information about their contacts. Through this exchange, the list of friends of friends (one social hop) of that contact is obtained. However, as both users also have information regarding the contacts of their other friends (acquired similarly), the peers continue the procedure by exchanging those as well. This results in a list of contacts two hops away. This pattern is continued until a predefined hop distance.

As explained in Section 2.9, most variants of Bloom filters can be combined (creating a union of all entries) by a simple bit-for-bit OR operation. Therefore, instead of storing, and sharing the multi-hop Bloom filters separately, we simply combine all filters at the same hop distance that we have received. By keeping these levels of filters separate, we can check not only whether a previously unknown identity is somehow connected to us, but also at which distance in our social network. Figure 7.1 illustrates this process.

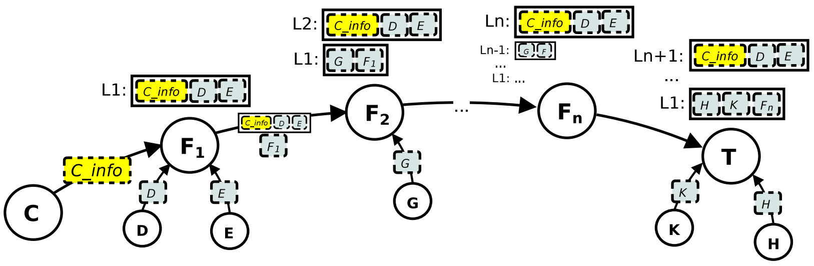


Figure 7.1. Viral propagation of relationship Bloom filters. For each hop, C's information gets pushed into a higher-level filter.

The simplest way of using these Bloom filters to spread relationship information is to simply define a standard filter size and add the identity of each contact (their public key) to the filter. However, this raises both security- and privacy concerns, as discussed earlier. Scalability is also an issue. With each passing, the number of elements in the filters may grow exponentially, easily overfilling filters designed originally for only a single node's contacts. In the following, we address these issues, enabling secure and trustworthy viral distribution of relationship information.

7.3.1 Threat model

Before describing the scheme, we define the security threats of a loaf-like, simple scheme in order to assess how successful we are in overcoming those. The goal of the scheme is to discover a social link between a connection establishing (or calling) party, and the target (callee) in a secure manner without compromising the privacy of either. For our model, we inspect the threats from the point of view of what a malicious peer belonging to each of the different types of actors within this scenario may hope to gain. These are the following:

Caller. As the primary purpose of the scheme is to filter connections based on social trust, the caller has the most to gain from being dishonest. The primary goal for a malicious caller is to be seen as much closely connected to the target than he is. As noted, in an environment where the identities are self- generated (whether signed by a trusted third party or not), it is easy to generate identities until one matching a Bloom filter is found, falsely identifying the caller as belonging to a friend's contacts.

Callee. The callee, although generally considered only as the potential victim of an attack, may also misuse the scheme by luring users into connecting to it. As it is the caller's duty to prove the relationship between the two, the callee will obtain information during the handshake. Unless the protocol is well designed, the caller might reveal too much. This includes information that can be used to gain other users' trust, or in performing off-line analyzes of the social environment.

Intermediates. An intermediate is a peer that is a member of the social link between the caller and callee. In general, we consider these trusted with regards to correctly propagating the Bloom filters. After all, the scheme is built on leveraging the trust for our contacts (the intermediates) in order to form new relationships. As any filter passing through intermediates will always be one social hop further away than the intermediate, there is no motive to alter the filters passing through, as they will never be trusted more than the intermediate himself. However, there is a privacy concern with handing over our filters to these. Even though we trust our contacts to propagate the filters correctly, we do not want them to be able to analyze the content (i.e., discover our other contacts).

Outsiders. An outsider is a peer that is neither an end-point of the connection, nor part of any social link between the two. The main threat of the outsiders, in our scenario, is to be able to analyze the structure of the social network from publicly available information.

We should note that these roles are often shared. For instance, the callee and caller may possess the same information as intermediates, and therefore be considered as posing the same threat (discover our other contacts). And each of the three other types can also act as outsiders. These categories merely reflect different positions peers can be in, and what they may strive to unlawfully gain. Furthermore, these specify only the roles within a one-directional authentication. For a complete authentication, we would employ mutual authentication where the roles of the callee and caller are exchanged.

7.3.2 Relationship keys

The first step we take is to prevent outsiders and intermediates from being able to dissect and analyze the contents of the filters (and therefore the structure of the social network) by obfuscating the entries (contacts) put into them. As noted, this analysis is possible if the public, well known, identifiers of users are used. In order to prevent this, we replace the public identifier with a value seemingly without any correlation to the identity, but that can be proved by the owner of the identity to be tied to it. As we are using public key- based identities, we use an electronic signature of the identity key. This value is impossible to construct without the private key of the signature, but can easily be verified and tied to the identity.

Although this prevents intermediates from arbitrarily dissecting the Bloom filters to find well-known public identifiers, it is only assuming that the intermediates themselves are not friends with those contacts. The signature value should be regarded as a secret, which users should only give out to people that they trust. However, as keeping a shared secret amongst all contacts is hard, users can create a set of relationship keys which are used according to the type of relationship. For instance, one key could be dedicated to close, trusted, friends, while another for mere acquaintances. During connection establishments, the caller would only need to prove that he is the owner of one of the keys that appear in the filter by providing a signature made by that key over his identity key, as well as the public part of the signature key.

If the filters a specific callee trusts are well known, or otherwise revealed, this scheme does not efficiently prevent users from engineering public keys whose signatures of an identity fit into a specific filter. We can enhance the scheme by inserting additional values to the filter that tie the key used to it more strongly. For instance, we could add the key itself. Finding a false positive key in that model can be compared to finding two interlinked random values that both pass as members of the Bloom filter. This would decrease the probability from the false positive probability p to p^2, meaning that for a filter with an error probability of one percent, an average of ten thousand keys have to be generated to find a match. Although the scheme can be extended even more (e.g., by adding signatures of signatures), it would also increase the amount of data, and still remain breakable given sufficient computing resources. We need a more efficient method to decrease the probability of successfully generating a matching key.

7.3.3 Detecting relationship-specific keys

Although the relationship keys prevent outsiders from generating identity keys matching the same Bloom filter patterns as specific well-known users', it requires that either the caller reveals his relationship keys, or the callee the filters he trusts. As discussed, the caller can decrease his dependency on a single key by using a set of keys, or even relationship-specific ones, which are revealed only when needed. The problem is how to know which ones to reveal to whom; assessing in which of our contacts' network a specific user might reside. We need a scheme that allows the communicating parties to compare and discover keys that they may have in common, without either one revealing too much. This we achieve using Bloom filter intersections.

Instead of the calling party sending the key signatures in plain text, they are also added to a Bloom filter, which is given to the callee. By sending the signatures in a Bloom filter, the calling party does not need to reveal the keys directly, but rather in a anonymized, probabilistic manner. As the filter contains multiple entries (possibly even false entries, *decoys*), it can not be used to discover the Bloom filter pattern for the individual entries, as it is impossible to know which bit positions relate to a single entry. The callee is therefore unable to discover a Bloom filter pattern for any single one of the caller's keys, to be used to construct a fake relationship key in order to gain trust with other users.

However, the purpose of this filter is not to single out a specific key signature that might be present in the remote user's relationship filters, but instead limit the scope. The remote user uses this filter to perform an intersection over the relationship filters he trusts, and sends back the result. The caller can now check which entries are still present and select one of those keys to present to the callee. As with the previous schemes, we can lower the false positive rates using multiple values of the same key (adding signatures of signatures), lowering the probability that a relationship key is falsely identified as being in the trusted filters. It is, of course, possible that more than one of the caller's relation keys exist in the callee's filters, in which case the caller selects one to be used.

The caller does, in any case, need to finally reveal a relationship key present in the intersection linking him to the callee. A dishonest callee could therefore forge the intersection by simply returning the Bloom filter containing the keys untouched (signaling that they all match). This would prompt the caller to reveal one, even though he is not linked to the callee. The dishonest callee would subsequently use that key to analyze the structure of the social network, or to create a new key matching the same Bloom filter pattern and thereby gain the same social status as the caller. However, this can be prevented using *decoy* keys. By creating a number of temporary, throw-away, keys, not used in any relationship, and inserting those into the filter passed to the callee, the caller has a range of keys to present to dishonest callees without revealing any of the real relationship keys.

This still does not solve the problem of the caller creating specially engineered keys to fit the remote party's trusted filters. In fact, by returning the intersection of the key signatures and the trusted filters, the remote user reveals a subset of the ones he definitely trusts. Even though the trusted filters are kept secret, the caller will gain insight into those. Fortunately, there are a couple of mechanisms we can use to minimize this leakage. First however, we need to impose strict limits on the false positive probability (*health*) of the caller's key filter. By limiting the false positive probability of this Bloom filter, we efficiently limit what can be deducted from the result of the intersection, as explained in more detail in Chapter 7.3.4. In addition to this limit, we use partial intersections and key escrow.

With partial intersections, we divide the relationship filters into two separate parts; a *public* part which is used for the intersection, and a *private* part which is used only for verification. When establishing a relationship, both parties share the public part of a key they associate with the relationship, together with a signature of their own identity key, made with that relationship key. The relationship key is added to the public part of the relationship filter, while the signature to the private part. During connection establishment, the caller creates a Bloom filter composed of all the relationship keys he has used (along with any number of decoys), which is passed to the callee who intersects it with the public part of the relationship filters he trusts. After the intersection, the caller will discover which of his relationship keys may be present in the callee's filters. If one is found, it is presented to the callee, along with the signature over the caller's identity key. The callee can now verify that the caller is linked by checking the private relationship filter for the inclusion of the signature.

Although the scheme requires the maintenance of two separate filters for each hop, the total amount of data that has to be distributed (the relationship filters) will not increase. Compared to using a single filter with two different values per key (the relationship key and the signature), we would need to double the filter size in order to see similar false positive probabilities as we see when using two separate filters. Therefore, as explained in Chapter 7.3.4, the total amount of data will remain the same.

Another mechanism the callee can employ to prevent the caller from opportunistically generating, or choosing matching, off-line generated, relationship keys after receiving the intersection is to require the caller's relationship keys to be escrowed before returning the intersection result. The callee simply asks the caller to provide all of the possible relationship keys, each encrypted with a different unrelated encryption key, in advance. After identifying a match in the intersection result, the caller will point out which of the keys was found, and provide the decryption key (along with the signature) so the callee can verify the claim. The complete process is illustrated in Figure 7.2.

7.3.4 Security analysis

As Bloom filters are probabilistic data structures, the security of any system built on these will inherently display similar, probabilistic, characteristics. As explained in Chapter 2.9, the probability of a fault (false positive) depends on the structure of the Bloom filters, and the amount of data contained. Our goal is therefore to clarify how these parameters affect the overall security of the BloomBuddies scheme, in order to define the minimum requirements for using it safely. We examine the security of the BloomBuddies scheme according to the threat model presented in Chapter 7.3.1.

As described in Chapter 7.3.2, the threats posed by intermediates and

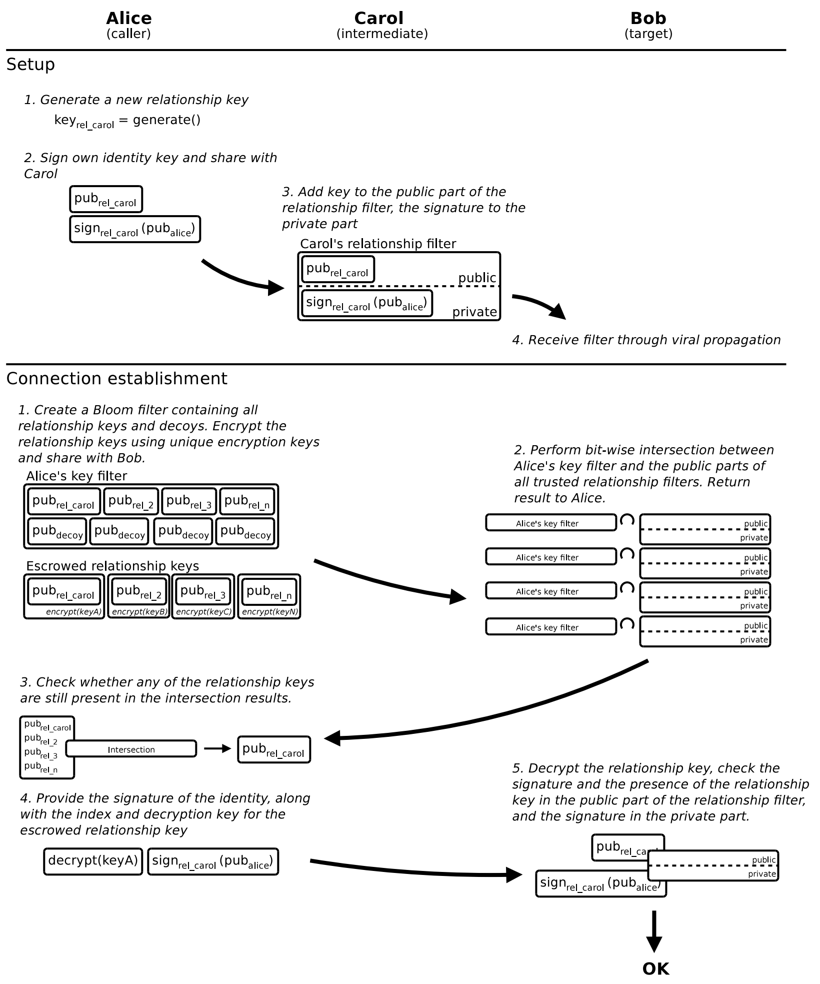

Alice
(caller)
Carol
(intermediate)
Bob
(target)
Setup
1. Generate a new relationship key
key$_{rel\_carol}$ = generate()
2. Sign own identity key and share with Carol
pub$_{rel\_carol}$
sign$_{rel\_carol}$ (pub$_{alice}$)
3. Add key to the public part of the relationship filter, the signature to the private part
Carol's relationship filter
public
private
4. Receive filter through viral propagation
Connection establishment
1. Create a Bloom filter containing all relationship keys and decoys. Encrypt the relationship keys using unique encryption keys and share with Bob.
Alice's key filter
pub$_{rel\_carol}$
pub$_{rel\_2}$
pub$_{rel\_3}$
pub$_{rel\_n}$
pub$_{decoy}$
Escrowed relationship keys
encrypt(keyA)
encrypt(keyB)
encrypt(keyC)
encrypt(keyN)
2. Perform bit-wise intersection between Alice's key filter and the public parts of all trusted relationship filters. Return result to Alice.
Alice's key filter
public
private
3. Check whether any of the relationship keys are still present in the intersection results.
Intersection
4. Provide the signature of the identity, along with the index and decryption key for the escrowed relationship key
decrypt(keyA)
5. Decrypt the relationship key, check the signature and the presence of the relationship key in the public part of the relationship filter, and the signature in the private part.
OK

Figure 7.2. The BloomBuddies scheme illustrated. Alice needs to prove that there is a social link between her and Bob through one or more intermediates (Carol).

outsiders are prevented by using specifically constructed relationship keys instead of identity keys as the Bloom filter entries. Even though this prevents the detection of specific users in a particular filter (within a certain probability), it is still possible to gain an understanding of the overall social network by reviewing and comparing filters. If relationship keys are re-used between different contacts, it is possible to compare different filters and approximate how many common contacts these contain. Furthermore, the size and fill ratio reveals the number of entries in a filter, which corresponds to the number of contacts of a person. However, as the false positive probability of a filter depends on the ratio between the filter size and number of entries, by increasing the size of the filters we are able to add false entries without compromising the security of the system.

We can therefore prevent size analysis by defining a standard number of entries each user inserts into their filter, whether they are real entries or not.

However, as the purpose of the BloomBuddies scheme to promote trust in the caller, our main concern is that the caller would be able to gain a better standing than deserved with the callee. If we assume that the system is otherwise secure (i.e., the relationship filters and -keys are not leaked), the most straightforward approach is for the caller to find a public key pair that is falsely identified as present in the callee's relationship filters.

By dividing the relationship filters into two parts, both the public key and the signature of the caller's identity, would need to fit into two separate filters. As discussed in Chapter 7.3.2, within our context, these can be thought of as two linked random values. The probability of finding such a key pair is therefore the product of the probability of matching two random values into these filters. With $m_{pub}, k_{pub}, n_{pub}$ representing the size, number of hash functions and entries of the public part of the relationship filter, and $m_{priv}, k_{priv}, n_{priv}$ respectively the parameters for the private filter, the total probability of finding such a key becomes:

$$p = (1 - (1 - \frac{1}{m_{pub}})^{k_{pub}n_{pub}})^{k_{pub}} * (1 - (1 - \frac{1}{m_{priv}})^{k_{priv}n_{priv}})^{k_{priv}} \tag{7.1}$$

As each relationship will insert one entry into both parts, we can assume that the number of entries, and the filter parameters (size, hash functions) are the same for both filters. The probability becomes then:

$$p = (1 - (1 - \frac{1}{m})^{kn})^{2k} \tag{7.2}$$

This means that the probability of being falsely identified as a member of a relationship filter (both the false key and its signature matching) is the square of the filter's two components (public and private parts). For instance, with an upper limit of 1% false positive probability on the components, the total probability is 0.01%. In other words, the caller would need to, on average, perform 10000 connection establishment attempts, with different keys, in order to succeed. We can compare this probability with having both entries (a total of $2 * n$ entries) in the same filter of size m_2. The false positive probability (for one entry) will then be:

$$p_2 = (1 - (1 - \frac{1}{m_2})^{k2n})^k \tag{7.3}$$

As we require that two interlinked values are falsely identified as present, the error probability will then be p_2^2. In order for this to be equal to the error probability of using two separate filters, we need to solve the size of the filter (m_2) from the following equation:

$$(1-(1-\frac{1}{m})^{kn})^{2k} = (1-(1-\frac{1}{m_2})^{k2n})^{2k} \tag{7.4}$$

Solving m_2, we have:

$$m_2 = \frac{1}{1-\sqrt{1-\frac{1}{m}}} \tag{7.5}$$

Which converges towards $2 * m$, meaning that m_2 will need to be two times m for the same error probability. Therefore, having two different filters requires only as much total data as maintaining two values in the same filter. The benefit of using two-part filters is to limit how much the caller is able to learn from the intersection result that can be used in subsequent attempts.

The callee will naturally strive to reveal as little as possible, preferably only the entries that legitimately correspond to relationship keys owned by the caller. The goal is therefore to minimize the number of bits set in the intersection result, in other words, the false positive probability. For an entry to be present in the intersection, it has to be present in both of the filters of the intersection. The false positive probability of the resulting filter is therefore the product of the probability of the two source filters. We can therefore limit the false positive probability of the result by adjusting either one of the sources. As the caller-provided key filter is composed during the connection establishment, it is the natural target for this adjustment.

Even though the callee is in this way able to control the false positive probability of the intersection result, it does not prevent an attack. Even though the intersection result would have an extremely low false positive error probability, with enough computing resources it is still possible to engineer a key matching the result. Furthermore, with each subsequent connection attempt the caller is able to further familiarize himself with the public part of the callee's trusted filters. It is the private parts of the filters that ultimately provide verification of the relationship.

The purpose of the public part of the filter is, as explained in Chapter 7.3.3, to allow the caller to choose which key (or keys) to reveal, as well as to discover rogue callees through the use of false entries. As described, the false positive probability of the intersection result is the product of the probabilities of the source filters; key filter and the public part of a trusted filter.

$$p_{\cap} = p_{keyfilter} * p_{public} \tag{7.6}$$

Assuming healthy filters (for instance, 1% false positive probability per filter), for a relationship key to erroneously to be indicated as present in the intersection result is low (0.01% in our example). Having more than one of the relationship keys present in the intersection result is certainly possible, but having any of the false ones is always due to the false positive probability as these keys are not used for any valid relationship.

Considering a rogue callee, the only way in which he will succeed in luring the caller into revealing a relationship key is for the key to be present in the intersection result. As the callee does not know which bits of the filter belong to valid relationship keys, the intersection filter will have to be produced at random. This means that the relation between the probability of a valid relationship key to remain in the filter versus a false will follow the relationship between the amount of valid keys in the key filter and false ones. Therefore, by adding a large amount of decoys, for instance ten times the number of valid relationship keys, the probability of finding decoys in these rogue intersection filter will be much higher than encountering a valid relationship key, independent on how populated the filter is. In our example, for each valid relationship key found will, on average, entail that ten false ones are found as well. With the intersection false positive probability of 0.01%, the probability for a rogue callee to succeed (i.e., produce a filter with a valid relationship key present, but no false entries) is $p_{\cap}^{10} = 0.0001^{10} = 10^{-40}$, in practice none. However, we should acknowledge that the presence of a false entry does not automatically rule the callee as dishonest, as the probability of finding these will directly follow the false positive probability of the intersection result. Instead, the process should be re-iterated with a new set of false entries.

The process does not prevent a malicious callee from performing a *man in the middle* (MIM) attack between two users who are known to have a relationship. As the malicious callee will use legitimate intersection filters provided by the real callee, the caller will become convinced of the relationship and reveal the relationship key. In order for the malicious callee to fabricate a relationship with the legitimate callee, he would then have to either choose to rely on the false positive probability of the private filters, or engineer a key that both fit the intersection result (the public part of a relationship filter) and whose signature is guaranteed to be present in the private part of the relationship filter.

As the malicious callee can not access the private part of the relationship filters, this signature would need to have the exact same Bloom filter pattern as the caller's relationship key signature. Even though it is possible to generate such a key, the amount of work needed is magnitudes more than fitting a key into even the most strict key- or relationship filter. Essentially the false positive probability of a filter with only one entry. Furthermore, in order to protect the caller, we can still employ mutual authentication where the callee is forced to prove its relationship with the caller.

7.3.5 Scalability

In order for Bloom filters to be efficient, we need to have a fairly good estimate of how much information they will carry. However, in the decentralized environments we target, the assumptions regarding the amount of social links we need to convey using the BloomBuddies scheme may vary from person to person. This makes it hard to define a standard size for the BloomBuddy filters. As Bloom filters do not allow us to merge, intersect and compare different sized filters in the way required, we need a scheme that allows us to scale the filters without loosing these basic properties, in order to allow participants to choose the size of their filters themselves.

As discussed in Section 2.9, several schemes have been proposed that increase the filter size dynamically. However, these are based on the assumption that we only need to know whether individual elements are present and does not allow us to compare complete filters, as required in our scheme.

We base our method for scaling on the use of modulo for mapping the bit positions, and the observation that the growth of the false positive probability for each addition depends on the filter size. Adding the same elements to a larger filter increases the fill ratio less than adding them to a smaller one. Therefore, by adjusting the filter size, even though the fill ratio and false positive probability remain the same, we can preemptively slow down the *growth* of the false positive probability. This will provides us with larger filters, which in the end will allow us to insert more elements while keeping the error rate at an acceptable level.

To be able to increase the filter size without access to the source elements, we use a budding approach; we create clones of the filter and append these until the required size is reached. The Bloom filters we use are essentially bit fields where each set position corresponds to the modulo of the hash values of the elements. As these depend on the size of the filter, we need to ensure that the values get repositioned where they would have been inserted in a filter of the new, expanded, size. But as we do not have the source elements (only the modulo of the hash), we cannot determine the correct bit locations for an arbitrary filter size. However, as the values are the remainder of the hash values divided by the filter size, by multiplying the filter size with a positive integer, we can calculate the potential positions where they could reside.

As we double the size of the divisor (the filter size), the remainder stays unchanged if the dividend is less than the original divisor. In case the dividend is between the original and the new divisor, the remainder will increase by as much as the value of the original divisor. Continuing further, if the dividend is between two and three times the size of the original divisor, the remainder will stay unchanged. We can see that the remainder will always either stay unchanged, or increase with as much as the size of the original divisor. Similarly when multiplying the divisor by three, the remainder will be either at the same position, increase by the size of the original divisor or increase two times the size of the original divisor. Therefore, by duplicating the bit field into the new space, we are sure to include the modulo of all of the elements with respect to the new filter size, as depicted in Figure 7.3.

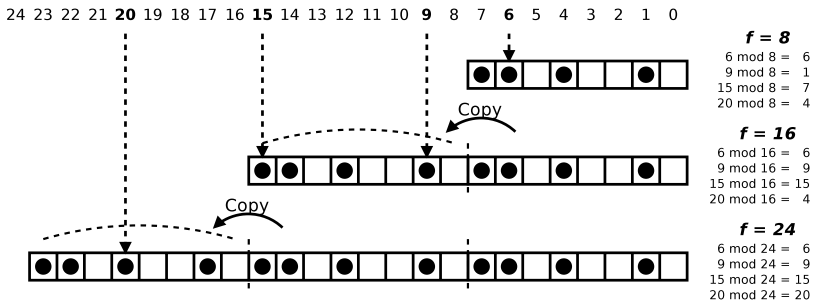


Figure 7.3. Increasing the size of an 8-bit filter containing the values 6, 9, 15 and 20. Each expansion appends a copy of the original filter.

In addition to providing a way of dynamically adjusting the filter size as the number of entries increase, more importantly, the scheme allows us to create unions and intersections of filters of different sizes. As we set each of the potential bit positions for the elements, the intersection will contain all elements that are present in both filters. And as the false positive probability of the resulting filter depends only on the false positive probability of the source filters (which remains unchanged throughout the budding process), it is not affected.

As intermediate nodes are now able to scale the filters they process, users can choose to use different sized filters according to their own assumptions and estimates of the maximum number of entries. This allows for flexibility in the network, as we do not need to define a global, cautiously chosen, standard for the size of the filters. This would likely be too large for most users, diminishing the efficiency sought from using Bloom filters in the first place. Furthermore, we are now able to define false positive probability limits on the filters created by a remote peers (the key filters and one-hop relationship filters) independent of how many entries they will contain. As the size of these filters no longer matters, the remote peer can choose a suitable size that will fit all the elements while maintaining the required error probability.

7.4 Implementation

The relationship-based filtering was implemented in the prototype as an access control filter (as mentioned in Chapter 3.3.1) for the SIP module. This module performed filtering of incoming sessions (voice and video call, instant messaging) based on the *social distance* between the caller and recipient. The module used the trustman module's services to acquire this relationship. Support for both the Pathfinder and BloomBuddies were implemented.

The BloomBuddies scheme was implemented using power of two- sized Bloom filters with an initial size of 32 KB (262'144 bits), and five independent hash functions. The default false positive probability limit of the implementation was set to 5% (i.e., the limit for considering filters *healthy*). The optimal value for the filters' parameters (the size of the filter and number of hash functions to use) depends on the expected number of entries, as explained in [169]. We based our values on the social networking study presented in [173], which found that a user with an average of 100 friends has, on average, 40'300 non-unique friends-of-friends. Our configuration represent the optimal parameters for filters containing this amount, while maintaining an estimated false positive probability of

4.4%.

The implementation followed the BloomBuddy scheme described previously. A relationship was formed by creating a new public key pair, of which the public part and, signature of our own identity key, was shared and stored in two-part filters. Four levels (social hops) of filters were used, meaning that three levels were exchanged during each update, resulting in 192 KB of data (two filters per level), when adhering to the initial filter size. However, at each update the filter size is re-examined by estimating the final fill rate per level using the average of the filters acquired. This affects the size peers instruct their neighbors to use for the filters they provide, in order to maintain a maximum of 5% false positive probability.

During connection establishment, the callee requests a key filter with a false positive probability of at most 1% from the caller. The caller constructs this filter calculating the optimal size for a filter containing 100 times more keys than he has shared, and subsequently adds 100 *fake* entries for each valid one to it. After the intersection, the result is checked for these fake entries, and a re-iteration (with a new set of entries) is performed if one is found. The connection is dropped, as a precaution, if fake entries is found after the second intersection. Otherwise the first relationship key found (if any) is provided, together with a signature of the caller's identity key. The callee compares these with the private and public parts of the relationship filters, and in case a match is found, provides the hop level of it to the access control module.

Reviewing the security analysis of Chapter 7.3.4 using the default configuration of our implementation, we can conclude that the scheme provides a reasonably trustworthy method for discovering social connections. Using the terminology of Chapter 7.3.1, outsiders are unable to gain insight into the social network, as explained previously. However, intermediate nodes are able to estimate the size of the social circles they are connected to, but as we are using unique keys for each relationship, cannot compare the filters in order to find common acquaintances.

As for malicious callers, the probability of finding a public key matching the relationship filter at a specific level is, at most (when the filters are filled to the 5% limit), $p = p^2_{relationshipfilter} = 0.05^2 = 0.0025$, or 0.25%. However, after the first attempt an intersection of the received filters can be made, and keys matching it produced. The probability of succeeding will eventually be limited only by the false positive probability of the private part of the relationship filter. As it is at most 5%, it would require an average of 20 attempts. We should note that these figures are only for the most crowded of the relationship filters, usually the most distant (4 hops). The filters containing more closely related contacts will, in most cases, have significantly lower false positive probabilities as they contain several orders of magnitude less entries.

Although an attacker performing a MITM- attack will be able to expose a relationship key used by the caller, the damage will be contained to only one specific relationship key. As for gaining trust with a legitimate callee, an attacker must rely on the same false positive probabilities as a malicious caller, or engineer a key matching the same Bloom filter pattern as the caller's. As we are using 32 KB filters and five hash functions, the probability of finding one is $p = (1 - (1 - \frac{1}{262'144})^5)^5 = 2.5 * 10^{-24}$, or 2.5 per septillion attempts.

The relationship-based filtering was applied to VoIP and instant message sessions. The user could configure the maximum social path length which he was willing to accept sessions from. In practice however, the graphical user interface (GUI) was designed so that one of four settings could be used; *Everyone*, *Friends only*, *Friends of friends* and *In your network*. *Everyone* indicates that all connections are allowed while *Friends only* and *Friends of friends* allow only the ones made by people close to you. *In your network* is designed to allow anyone that has some sort of social link to you. We review the usability of our solution in Chapter 8.

7.5 Summary

We have presented a Bloom- filter based method for discovering social connections between users. By using viral distribution and a novel protocol for exchanging filters, we can safely discover links of up to a predefined distance. Compared to existing solutions, our model is fully distributed, protects the privacy of both parties and reduces the possibility of cheating efficiently. Although we have presented it as a mechanism for discovering common acquaintances, it can be used to discover links between any type of information, as long as the two parts of the exchange are able to implement the interactive discovery protocol.

However, the BloomBuddies scheme is not perfect. It is based on probabilistic data structures, which means that there is a possibility for false identification given sufficient time and computing resources. It should therefore be used carefully, together with measures to prevent automated attacks. To our knowledge, it is the first scheme that solves the introductory problem in fully distributed environments, addressing the privacy and security concerns of all parties involved.

8. Usability

The underlying theme of this dissertation is to examine how networking services and applications can be constructed to operate in a *secure* manner, while not being reliant on a centralized trusted provider. The model we have outlined implements privacy protecting methods and utilizes technologies such as the Host Identity Protocol to secure the communication and, if used properly, to minimize the risks involved.

However, due to the fundamental differences in how the services are provided, the threats may not be immediately understood by the end users. In order to provide real security and privacy, users should have a basic understanding of the threats and be able to properly use the security technology. In practice, poor usability has often been more detrimental to system security than the weaknesses in the underlying security mechanisms [149]. Users are unable to detect security indicators, and demonstrate click fatigue when running into security warnings [150]. Even when a user sometimes does take the time to look at the security indicators and consider the warning, he may still fail to interpret and utilize this information correctly [38]. This leads to users falling prey to relatively simple social engineering attacks [15].

Despite the risks associated with Internet- based services (e.g., VoIP), some users prefer these over traditional alternatives for their easy availability. In P2P- based variants, the security issues become even more tangible. As previously reviewed, P2P systems harness the shared resources of (possibly malicious) end-users to provide the service, creating even more opportunities for exploitation The additional security mechanisms we have presented are designed to mitigate these risks, but need also to be presented to the user in a way that is both understandable and usable. This is even more challenging with the demands and constraints of the mobile environments we are targeting.

User studies are needed to understand how, when and what to communicate to the users about these security mechanisms and the threats involved, in order to understand how these should best be managed. In this chapter we provide a comprehensive look into the usability challenges of our system through the use of a P2P VoIP application on a mobile device implementing the additional security features. We have gathered user feedback via interviews, an online questionnaire and a UI paper mock-up. We present and analyze the data gathered on users' views and attitudes towards the emerging application and on Internet telephony in general.

8.1 Related work

It is commonly believed that the challenges in making security usable markedly differ from the demands of so-called traditional usability [176] [149]. A key difference is that few, if any, errors can be allowed not to jeopardize the overall security. This means that the common trial-and-error way to learn the use of a new application will not work learning how to manage security.

Security and usability are often seen as competing goals – a classic example is usability of passwords: a password that is easy to remember is weak, and strong passwords are hard to remember [15] [38]. Security and usability also compete in another way: in [7] the authors found that users often give up security for easy access. In other words, if security becomes an obstacle in conducting everyday tasks, it gets turned off [124]. Apart from the competition between the two, also bad usability hinders security management: users may even be aware of the risks based on their experience, familiarity with the domain or sender, and of the presence of security features, but are unable to make appropriate security decisions due to bad usability [38].

When it comes to usable security for VoIP and more specifically P2P VoIP, not much is known about the current usability issues or how P2P VoIP users behave and why, especially in a mobile environment. In [56] is provided some insight into the user behavior and motivations of Skype users, when they investigated how P2P VoIP traffic in Skype differs from traffic in P2P file-sharing networks or from traffic in traditional voice-communication networks. According to [56], there were some remarkable differences in the usage patterns of Skype users as compared with the usage of traditional P2P file-sharing systems such as KaZaA, which also have their problems with usability.

8.2 Research questions and methodology

Our research concentrated first on finding out about relevant user habits that are likely to affect the usage of P2P VoIP application on a mobile device, and what the current level of understanding related to the security of such an application is. Our aim was also to find out any relevant user habits related to security in the context of using a mobile device, as these may affect what components a mobile user interface for presenting the security options should concentrate on. Our research questions (RQ) for this part were formulated as follows.

RQ1. Which current usage patterns (computer, mobile phone, information security) are relevant for the usage of our application and, on a more general level, for managing secure P2P VoIP on a mobile device?

RQ2. What is the current level of understanding of security related to P2P VoIP on a mobile device? What do users use as reference point for understanding the security of the emerging application?

RQ3. What are the privacy needs in using P2P VoIP on a mobile device?

Furthermore, we wanted to collect feedback and impressions for proposals for the interface design for controlling the security features (presented in Chapter 5 and 7), which we then implemented as a P2P VoIP application using our prototype implementation of the P2P system. We were most interested in how users, who were not experts in security or in information and communication technologies, would be able to grasp the concept and successfully use the application. However, we still wanted our users to be fluent computer and mobile phone users – possible early adopters. The security mechanisms of the application are such that they require a decent understanding of the underlying networking concepts to be fully grasped. The target demographic of the study was therefore chosen so that the full details of the mechanisms (and thus of any unexpected behavior) would not be well understood, but the areas to which they relate, and the risks involved, would be familiar. This would allow us to study the layman's impressions of the issues of our study, without having to dwell into the precursory concepts.

We used a mixed-method approach to get several types of information on the user requirements and on the usability of the system: 1) semi-structured interviews, 2) an online questionnaire and 3) user testing of user interface mock-ups and 4) on a functional UI.

8.3 The study

We started our study with user interviews, in order to understand the current user attitudes and behavior relevant to our study. We chose to perform interviews, as it allows to probe more deeply into the topics, finding subtleties that are hard to discover with more fixed-form methods. At this point, we only had our research questions and a rough outline of the area, so we wanted to encourage the subjects to bring up any issue that he or she might associate with the topic. The aim was to gain a comprehensive understanding of people's attitudes towards mobile security, not only their views on the topics in our initial outline.

On basis of the outcomes, we created an online questionnaire of the most important issues, to gather more responses than is possible through interviewing alone to form a understanding of the general attitude towards these issues. We chose an on-line questionnaire, as it has been proven to be a cost-efficient technique for gathering data from a larger population. The goal was to quantify the concerns discovered during the interviews. The combined findings were then used to inform the design of the first paper mock-up user interface.

The mock-up was subjected to user reviews and feedback under moderated semi-formal usability testing sessions. The feedback gained on these sessions served as input for the VoIP application UI on a Nokia N810 Internet tablet, which was also subjected to user reviewing and feedback under moderated semi-formal usability testing sessions, where users conducted several test tasks with the mock-up, at the same time discussing their experiences and thoughts about the mock-up with the moderator. We will now present each phase of the conducted study in more detail.

8.3.1 The Initial Interviews

The initial interviews were conducted with 9 users, aged 18-25, 6 male and 3 female. Most had a bachelor degree, three in computer science. By the time of the interview all the participants had approximately 2-3 years of VoIP experience. However, none of the participants had ever used VoIP application on any mobile device. Interviews took 45 minutes each and focused on finding out about users' experiences and preferences that were likely to be related to usage of VoIP. We also inquired about users' general knowledge of both computer security and security in VoIP applications.

We found out that all had received unexpected contact attempts via a VoIP application, however most had rejected the contact attempt. The interviewees usually only interacted with their contacts via VoIP, but were unable to reliably estimate how many buddies they had on their contact lists and how frequently they used VoIP. Mobile telephony was the most frequent way to contact friends, but Internet telephony was quite popular too, to avoid long-distance call charges. Four interviewees mentioned dropped calls as an undesirable aspect of calling through Internet.

The interviewees exhibited *typical* security attitudes, considering themselves relatively knowledgeable about security yet unable to explain basic security related terms such as phishing. But the more the participant seemed to know about security, the less he considered himself to understand it, and vice versa: the more ignorant felt the safest. This finding on mismatch between estimated skill level and awareness on security could in our view be of crucial importance: falsely based carelessness can create an easy attack point. However, the interviewees realized on some level that there were more risks related to Internet calling, yet one user believed VoIP to be the safest calling method available.

Reasons for the general mistrust of VoIP were manifold: one user expressed he did not trust VoIP because "they [the service provider] can sell my profile info." Although this opinion affects the overall trust in VoIP, it shows lack of trust towards the provider rather than being afraid of an actual attack and is not as such related to the actual security of VoIP. In users' mind this worry was, however, part of the security concerns. This is an important finding as it is crucial for the UI design that it encompasses what users find as relating to security, even when from a technical viewpoint it might not be related to security. To trust or distrust may depend on such false assumptions and conceptualizations, and if ignored, may stop the user from using the system.

The interviews revealed that there are some reservations towards the security of VoIP, but concrete reasons were hard to pin down, as the understanding of the concepts varied. Therefore, in order to be able to describe any security mechanism efficiently, we sought to quantify how familiar different security-related terms are through the questionnaire, and if there are some that are misunderstood. Furthermore, as the trust in the service providers was an issue, we needed to assess how important privacy in VoIP is, which affects how security settings related to it would be received, and whether these settings could make VoIP be more attractive. Finally, in order to gain insight into the usefulness of the social filtering mechanism, and therefore how obtrusive dialogs related to it would be, we wanted to know how common unsolicited VoIP calls or messaging is, and how familiar people are with the associated dangers.

8.3.2 The Online Questionnaire

The link to the questionnaire was sent to several mailing lists and newsgroups to draw users. We also tried to gain respondents through Facebook by advertising the study on one researcher's personal Facebook account. The questionnaire can be viewed at https://survey.hiit.fi/index.php?sid=-57695&lang=en. With 44 questions, the questionnaire began with basic demographic questions (age, gender, education, nationality, native language), and then continued with questions on telephony usage in general: preference on the type of telephony used (mobile, land-line or Internet telephony), and on previous experiences on Internet telephony (if any) and on their current level of understanding of security: password practices, information sources for security, worries related to security and privacy and which applications they currently trusted. We then asked if they recognized some security related terms such as phishing and pharming. Finally, the last questions dealt with the respondent's reactions to previous connection attempts, if any, from unknown callers or chatters over a VoIP application. This part is relevant for the P2P VoIP calling, where such connection attempts present a realistic use case.

Many questions were conditional, with the aim of having respondents only go through questions that were relevant to their experiences: if a respondent had used Internet telephony, the questionnaire would consist of questions on the usage habits: how often, with whom, for what purposes, and which applications the respondent was using for Internet telephony. If a respondent had never used Internet telephony, she was presented with questions on why not.

The questionnaire ended with a request for contact details for a possible invitation to test "a new user interface for a mobile VoIP application" – we recruited our mock-up testers among the questionnaire respondents. If the respondent did not enter their contact details at this point, they remained anonymous.

We received a total of 103 complete responses, 51/51 male/female (one respondent did not state gender). The majority of the respondents (43%) were 25-29 years old. The respondents came from 28 countries; their educational background was diverse. The data was analyzed against the research questions:

VoIP was the preferred calling method only by 12% of the respondents. Unlike the interviewees, 14% of the respondents had used VoIP on some type of mobile device (other than laptop): a typical usage was calling and chatting. 65% were interested in having VoIP services on their mobile devices, although often for infrequent exploratory use only. The most annoying feature of VoIP applications were delays, unexpected connection breakdowns, noise in the channel, and bad quality of voice.

The respondents seemed to associate security with passwords, viruses, privacy, confidentiality, and integrity of personal information. They seemed to be aware of computer security in general; 93% said they discuss issues related to security with other people. The majority was aware of at least one major system attack and had had personal experience of being infected by a virus as well. 32% believed they are very well informed about security. Only 6% believed to be badly informed about security.

On security related behavior, 42% stated that it was easy to remember their passwords; 81% did not write down their passwords while 71% were using same password for multiple services. The main security concern was loss of personal data, none was using a good password policy. Passwords were categorized into “simple” and “complicated”, in terms of their memorability and structure. The simple passwords were used for “non-sensitive” services such as University accounts – or e-mail accounts, which were somewhat surprisingly considered non-sensitive by many users. The more complex ones were for systems used for financial transactions.

The majority was unaware of encryption in VoIP applications. Only one participant wished for encryption in VoIP, and three felt encryption should depend on the nature of the call. Though the need for security seemed small, the respondents showed relatively strong concerns towards online privacy. When asked whether they believe their online activities can be monitored, 50% had wondered if their Internet calls could be listened to, and 60% had wondered if unauthorized parties could read their chat sessions. The percentage of users that never thought about being violated was 24% for calls and 21% for chats. Users seemed to believe that no one would even try to eavesdrop on their conversations: “Who cares to know what I'm talking about!".

The finding is in line with earlier work: most users tend to fall in the category "privacy pragmatists", who are aware, but relatively careless about privacy, ready to trade it off for a bargain [6].

In order to understand how users deal with privacy violations, we asked whether they had received via VoIP a call, or chat request, from someone not on their contact list. Almost one third had had at least one such a call and almost half at least one such chat request. 17% rejected the call and 14% also subsequently blocked the caller. For chats, 15% replied but 22% blocked the stranger.

We also attempted to analyze the amount of trust users have for online applications and services, and how they decide what to trust. 46% were neutral regarding trust in VoIP; only 3% were absolutely trusting and only 3% did not trust VoIP at all.

8.3.3 Creating and Testing a Paper Mock-up UI

Next, we crafted a set of paper mock-ups for the P2P VoIP application for gathering user feedback. The goal was to study how interface proposals for the security mechanisms presented in this dissertation would be perceived on a mobile device. Although the mechanisms apply to all services provided through the P2P system, the mock-ups (and later user interface) was focused on VoIP, as it provides a familiar use-case.

As the questionnaire had confirmed that privacy was an important issue for people, the paper mock-up UI covered use cases mostly related to dealing with unknown peers. These were:

Use case 1: User is calling

Use case 2: User receives a call

Use case 3: Introducing a buddy.

Use case 4: Managing your security/privacy settings

Figure 8.1 shows the main UI on the paper mock-up. Online and offline contacts were visualized with green (online) and gray (offline). The main UI also included personal information of the user (1), profile preview (2), dial pad (3), status (4) and the social distance mode (5). The personal information included the user's name, profile picture and a personal, editable, message. The "Profile preview" showed how other users see the profile. The dial pad was used for calls to people not on the contact list. The status could be set to On, Away, Invisible or Off, and the social distance mode was used to filter incoming connections based on how well the

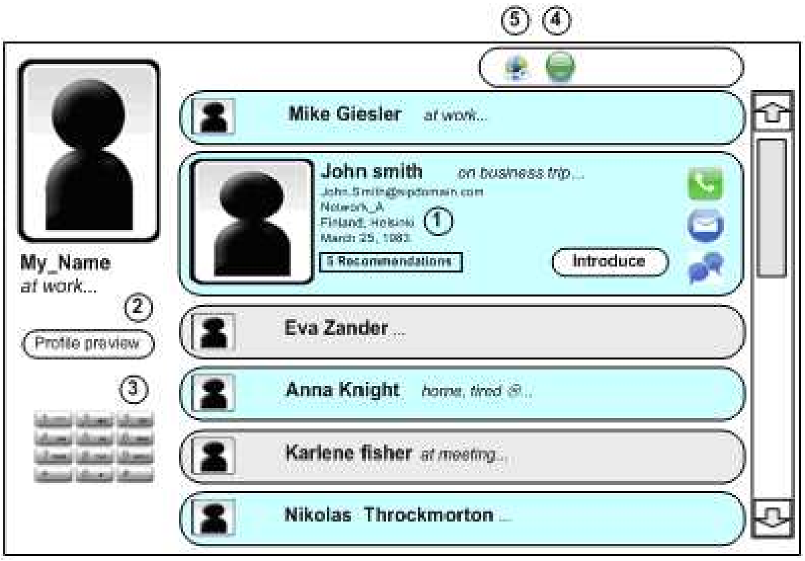
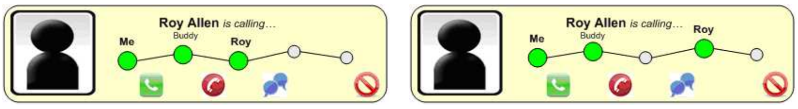


Figure 8.1. Application main UI paper mock-up.

user is connected to the caller.

Use case 1 and 2: User is calling and receives a call.

When receiving a call, a button with a question mark appeared next to the caller that, when pressed, displayed the social link we have to the caller. As examples of these information displays, we presented mock-ups such as Figure 8.2(a)-8.2(b), showing how the user is related to the caller.



(a) 2-hop connection with the caller. (b) 3-hop connection with the caller.

Figure 8.2. The interface presenting the user's relationship to the caller.

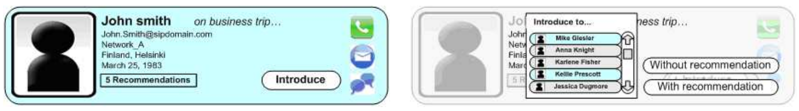
One central concept was *hops*, the social distance between two users in a network of buddies. We were interested in what number of hops would still increase the trustworthiness of the caller.

Use case 3: Introducing a buddy.

In our design, "introduction" was possible only from a direct buddy to another buddy. The introductions could be done with or without recommendation. A recommendation was given as free-form text (Figures 8.3(a)-8.3(b)).

Use case 4: Managing your security/privacy settings.

The default settings consisted of five parts: communication, modes, status, buddy list and recent. The modes allowed the user to change the

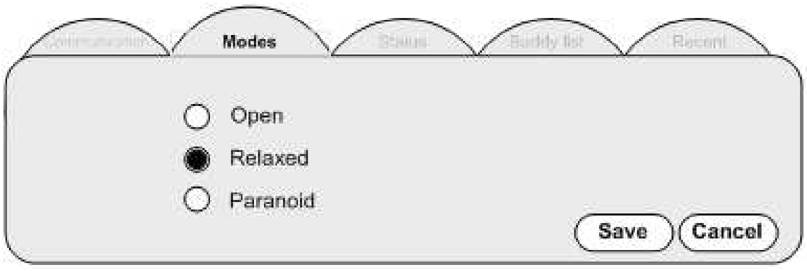


(a) Starting the introduction for a person in contact list.

(b) Choosing to introduce a contact with or without a recommendation.

Figure 8.3. Introducing contacts with and without recommendations.

privacy mode (Figure 8.4).



Open
[✓ Relaxed]
Paranoid

Figure 8.4. Choosing privacy mode.

8.3.4 Mock-up user test and analysis

We subjected the mock-up design to a semi-formal usability test where users were asked to observe and comment on the paper prototype against the four use cases. Users first filled in their demographic data and were then introduced to the use cases. Their reactions to the paper prototype were first observed and then discussed with the test moderator. Six university students between 18-24 years old, two female and four male participated in the study.

The main UI seemed relatively understandable: Users were able to match the functions with the right buttons. It was easy for users to interpret the various colors for buddies in the contact list. All users could identify correctly the buttons for calling, chatting and sending mail.

Use case 1 and 2: User is calling and receives a call.

The first two use cases were easily understood. In use case 2 (receiving a call), a question mark button was presented, which was designed to reveal more information about the caller. Almost all participants understood this. However, everyone did not understand how this information was obtained and found our UI solutions awkward. One user proposed that the icon be changed into a magnifying glass, in order to show that it would lead you to more detailed information; another proposed alternatively clicking on the profile picture of the caller to get more information.

The concept of displaying a common buddy was well understood, and liked. However, some users claimed they would answer the call independent of what is shown of the caller. Unconcerned with privacy, they stated that accepting a call from a previously unknown person would depend on their mood and current activities.

However, judging by the reactions to the mock-ups, it seemed that most users would not answer a call coming from a user further than one hop away: Four users stated they would only answer calls from at most a buddy of a buddy. One user would answer only to buddies' calls, and one user would answer a call even within four hops. An interesting suggestion for how to indicate your relationship to a person was to add a button with a number inside. The number would indicate the number of common friends. Pressing the button would reveal a list of these common buddies. Users could also prevent themselves from appearing on such lists by an option in the personal settings. User could also choose to be visible only as an anonymous "common buddy", without revealing their name.

Use case 3: Introducing a buddy.

The concept of introduction seemed easy to grasp. However, some users misinterpreted the icon. One user mistook "introduction" as a way to obtain additional information about the user; another user mistook it for an invitation to a chat session. One user also misunderstood the direction of the introduction. Finally, one user assumed introductions were possible only among buddies concurrently online, even though this was not the case.

Use case 4: Managing your security/privacy settings.

Although a new concept, most users seemed to understand that modes were somehow related to security and privacy. One participant however related the term "mode" to emotional states and mood, believing that this would affect the layout and look of the profile.

8.3.5 VoIP UI

Based on the feedback from the mock-up user tests, we created a status bar control menu, as well as a VoIP prototype application for the Nokia N810 Internet Tablet. The status bar control was implemented for the Internet tablets, which mimicked the native presence status control. In addition to the presence control (online status), we added two new items; the privacy settings and call filtering. These were implemented as sub menus to the control menu, allowing the user to set the privacy mode (Open, Relaxed and Paranoid) and the call filtering (Friends only, Friend of friend, In your network and Everyone), as illustrated in Figure 8.5.

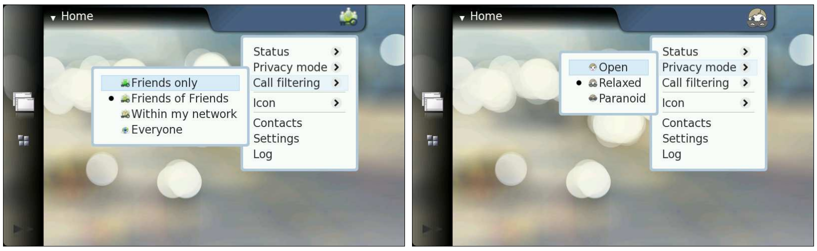
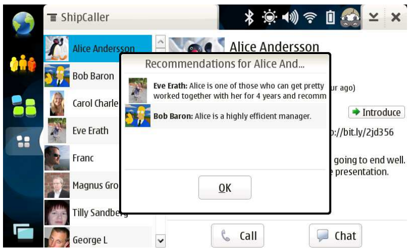


Figure 8.5. Snapshots of the Maemo status bar control interface. Both the privacy mode and the call filtering could be configured, although the status of only one of these were shown in the status bar icon.

The VoIP application provided standard instant messaging and voice calls between users, as well as the ability to send and receive recommendations and introductions. The application's main interface (Figure 8.6(a)) consisted of a contacts manager where detailed information about contacts could be viewed. This included images of the contact, simulated location- and social networking updates, and any recommendations given or received. The user could call, start a chat session or introduce (and recommend) the contact to other users. Based on the feedback from the mock-up tests, we clarified the direction of introductions through arrows, and emphasize that the functionality was available even though the contact was offline.



Recommendations for Alice And...

Eve Erath: Alice is one of those who can get pretty worked together with her for 4 years and recomm

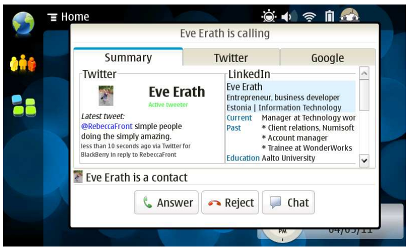
Bob Baron: Alice is a highly efficient manager.

ur ago)

p://bit.ly/2jd356

going to end well.
presentation.

(a) The main UI of the prototype application on Nokia N810.



Eve Erath is calling

Twitter

Eve Erath
Active tweeter

Latest tweet:
@RebeccaFront simple people doing the simply amazing.
less than 10 seconds ago via Twitter for BlackBerry in reply to RebeccaFront

LinkedIn

Eve Erath
Entrepreneur, business developer
Estonia | Information Technology

Current	Manager at Technology wor
Past	* Client relations, Numisoft
	* Account manager
	* Trainee at WonderWorks
Education	Aalto University

Eve Erath is a contact

(b) The extended call-notification view.

Figure 8.6. User interface for the P2P- aware VoIP application.

Upon receiving VoIP calls, an information panel was presented as part of the phone call alert (Figure 8.6(b)). It contained images, recommendations, the user's social connection to the caller, as well as simulated publicly available information (twitter tweets, search engine results) regarding the caller. This was designed to provide a richer experience than what the ’awkward’ mock-ups had offered, providing a context to the social link (i.e., through what sort of communities the caller might be linked to the callee).

8.4 Conclusions

In this chapter we have described how we have gathered user feedback through different phases of creating the application and its UI in order to make the emerging application and its unfamiliar security functionality as understandable and usable to its intended users as possible. The information gathered relates to current usage of mobile and Internet telephony, and their perceived security. At each new phase of the design, we used the previously gathered user feedback as input to guide the design to better reflect user needs for the emerging application.

Privacy, both in terms of protecting your communication, as well as knowing how to deal with unsolicited requests, is becoming more important each day, affecting the type of communication technology we choose. Our approach of utilizing social links has been well received, but the manner in which they are presented has an impact of its perceived trustworthiness. The interfaces we have created were considered a step in the right direction, and can easily be adopted to similar systems.

However, only usage over an extended period of time would reveal if the application enables security and privacy management that is both acceptable and also desirable for the end users, and what will the effects of habituation be on the actual usage.

9. Trials and future work

As part of the project, we conducted a trial of the prototype system within the networking research group at Helsinki Institute for Information Technology HIIT. We distributed approximately 20 Nokia N810 Internet tablets to our colleagues, with the encouragement that they would use the prototype system for P2P voice and video calls. The goal of the trial was to evaluate the system through hands-on experience. We hoped to gain feedback on both the concept itself, as well as the usability of our solutions and actual implementation. As these were distributed in an early phase of the development, we also expected the trials to help the implementation process, as a larger user base could reveal programming faults faster.

The first challenge was to execute a smooth deployment of the test environment. Instead of pre-installing all of the software and settings necessary, we wanted to evaluate methods could also be used by real users, outside of our trials, to set up their own network. This would provide insight into the deployment issues of the framework. Therefore, the only modification we decided to perform to the tablets was to upgrade the Linux kernel version to support IPv6 and other extensions required by the HIP daemon. This is an unusual and technical operation not commonly performed by end-users.

During our evaluation with HTTP-based applications, we often had to create different types of *wrappers* around the applications to make them work with the interface provided by the framework (described in Chapter 3.3.2). As examples of HTTP-based applications, we created a P2P file-sharing application and a personal photocaster. The client applications were often easy to configure. Most desktop web browsers can easily be set to use the HTTP proxy interface of the prototype, resulting in intuitive and memorable URLs for the user (e.g., *http://alice.at.p2pship.org/media*). The podcast client we used lacked support for HTTP proxies, but could use the URL-based interface. With the *server* software, we were required to bundle additional bootstrapping applications or -scripts, as these needed to be *registered* with the P2P proxy in order to serve requests.

SIP-based multimedia applications were much easier to use with the system. The design of the SIP ecosystem fits the framework much better, as registration of the clients is built into the protocol and these rely on the SIP servers to relay traffic for them. All SIP applications we encountered could be used unaltered with the prototype, just by configuring them to use the P2P proxy as a SIP proxy.

To ease the installation of the software, we set up an application repository containing the pre-compiled software packages. These are databases containing software that the graphical application manager of the tablet uses. Furthermore, the Maemo OS supports a mechanism of adding new software repositories to the application manager by opening specially formatted *repository* configurations files. We created one for our repository and distributed it through a web site as well as by email. This enabled the prototype system to be installed with a few clicks using the built-in application manager.

The next challenge was to configure the prototype system. As discussed in Chapter 8.3.5, the security settings of the framework were accessed through a status bar plug-in. The identities, however, still required manual initialization and importing to be accessible by the prototype. Furthermore, these would also need to be signed by a trusted identity authority in order to prevent impersonation.

The identity acquirement was solved by creating a web-based identity authority service site. The site featured a simple interface for requesting an identity and used email verification, limited to the *hiit.fi* domain, to tie the identity to a real user. The identities were signed by an identity authority created especially for the trials, and downloaded (after being reviewed) in XML-formatted data files with the suffix *.identity*.

The prototype software package contained in addition to the software a mime-type definition and a desktop integration description. The mime-type definition added a new file type to the Maemo OS, the *P2PSHIP identity package*. This was designed to identify the identity packages downloaded from the identity service (files with the suffix *.identity*), and were given the mime type `application/x-p2pship-identity`. The desktop integration description contains information about software, such as the type of software, the services it provides and how it should be categorized (e.g., a game, utility or graphics application). It also defined the content types which the application manages. By adding the newly created mime type as content type for the prototype, the identities would be automatically installed from the web site after being downloaded.

Although the trial environment is still active, we can at this point highlight some observations. It has helped us in improving the prototype, identifying faults and unexpected complications. The installation procedure can be considered successful, as each one of the participants was able to install the software and acquire an identity with low effort. It seemed that configuring the actual device was even harder: at the start of the trial, many experienced problems connecting to each other. It seemed that the users had formed two groups which could connect internally, but not with each other. The problem was found to be the time zone setting of the tablets. While initiating the device, it requires the user to configure the current time and date. What went unnoticed by many was that it also required you to specify the time zone. This resulted in some of the users using the default (GMT), while others had configured the correct (EEST). As these are two hours apart, the registration packets with a validity of one hour were either expired or not yet valid between two users of different time zones.

Another observation has been the importance of presence. As discussed in Chapter 3.3.1, we did not implement full presence support in the SIP module. This was a clear underestimation of its importance, as it may cause users to restrain from using the system (and seek an alternative communication channel), as they are not sure whether the user will respond. Finally, the power-saving techniques used by the tablets often caused problems. As it is a mobile, resource-limited device with a limited power supply, it has advanced methods of powering down components when not active. One of these is the WLAN interface and the networking stack. In centralized communication systems, the communication application periodically polls the service provider for updates, keeping a line open for incoming requests. In the P2P system, this is not always the case, which caused users to be unavailable unless they had engaged in some sort of network activity recently. This, however, can be addressed by maintaining artificial connections or actively participating in an overlay network.

9.1 Application experiments

The trial environment has inspired the development of a number of additional extensions and framework-aware applications that illustrate how the environment can used. In the following we provide a brief description of some of these.

ShipSharing. ShipSharing is a web-based social content-sharing application that utilizes the P2P HTTP interface of the prototype. The goal was to create a web-based content-sharing site that allows users to better control how and to whom the content is exposed. We created a small social networking site, similar to Facebook[1], that lets users establish friendships and interact with each other. The site featured a personal *wall* (writable note board accessible for the user and his friends) and a content-sharing page for each user. The site allowed content such as images, videos and documents to be shared with users that had been accepted as *friends*. However, the site used the P2P framework for actually delivering the content.

A content management application, implemented in the Python scripting environment of the prototype, was installed on the user's device. This provided content-sharing services through the P2P HTTP interface by featuring a generic interface that accepted requests for sharing content (opening a dialog allowing the user to choose the content to be shared) and accessing the content that had been shared. When adding content to the social networking site, it was not uploaded to the site itself. Instead, the site used the framework to issue a P2P request to the user's content management application requesting for new content to be shared. After the user had chosen the content to share (or rejected the request), the content itself was not transmitted, only an identifier used as index by the content management application.

The social networking site used this index when rendering the user's shared content page. As the site did not have possession of the actual content, it only linked it to the site by providing a reference to the content sharing application of that user. This meant that the actual content was fetched using P2P HTTP from the user when viewing the page, with the content management application allowing the owner to restrict to whom the content was shared. The user could set rules for how content was shared, which could result in some being shown while other content was

[1]http://www.facebook.com

inaccessible.

The experience we got from using the site was encouraging. Although the content sharing suffered from availability issues (users being offline or on slow Internet connections), the concept itself seemed to work. And as we only experimented with one application, we have not explored the full potential of such a system. The content-sharing application attached to the P2P framework provided, as stated, a very generic interface. The requests for sharing of content provided a parameter specifying the type of content to be shared. This enables the same application to be used for other sites (and content types) as well, which would in turn allow the user to manage the content shared throughout all these sites from one location.

We later ported the content sharing web application to the popular social networking site Facebook as well. This was possible through the Facebook application interface, which allows third-party applications to be used within it. Although the social networking is managed by Facebook, the content is transferred P2P, allowing users to better control with whom content is shared. The application is hosted at `http://apps.facebook.com/shipsharing`.

CoffeeSipper. The CoffeeSipper prototype consists of a custom SIP-based Internet video call application designed for the Internet tablets. When receiving a call, it automatically answers it and begins to stream the video feed from the built-in camera. It was originally intended as a playful P2P version of the Trojan room coffee camera[2], but can be used for any type of surveillance. Furthermore, support for SIP instant messaging has been added, allowing users to send messages which are shown on the display. The application is maintained at the SPEAR project repository[3].

DoorPlates. The DoorPlates application consists of a UI and a custom SIP client that is intended to replace the static room door plates (displaying the names of the inhabitants) seen in offices. By replacing these plates with an Internet tablet, we are able to provide additional, and more dynamic, information. For instance, the SIP presence of the inhabitants could be indicated on the UI. Status information can be sent directly from the inhabitant's P2P SIP application using an instant message to the door plate. When away, visitors could use the door plate to make a video call or leave a message, which is delivered as an instant message. The system is also maintained at the SPEAR project repository.

[2]http://en.wikipedia.org/wiki/Trojan_Room_coffee_pot
[3]http://code.google.com/p/p2pship

SIP Gateway

The SIP integration enabled the framework to be used with a range of real-time communication software, but only to communicate with other users of the P2P network. This was experienced as restrictive and inconvenient, as other applications were needed to contact friends that were not part of the trial. Furthermore, there may be communication systems that can not be converted to the P2P model, as they need centralized control for special security policies or billing. This led us to examine how the P2P SIP communication could interact and co-exist with traditional, centralized, SIP architectures. In more general terms, we set out to find how P2P real-time communication systems can co-exist with centralized ones. This would provide a more realistic user experience, as we are able to use the same application for all our communication, whether P2P or not.

Interaction with an external SIP system was designed using a gateway architecture. The SIP signaling is routed to a *gateway adaptor* plug-in of the prototype, which forwards it to an external SIP server. Although each user can act as a gateway itself, the gateway may also reside on a remote peer. This might be useful for corporate environments, where the management of the external accounts is centralized to a stable, well-connected and -protected, peer server.

The gateway functionality was implemented in the prototype using redirection rules. When receiving SIP signaling, the target of the message is compared against a set of regular expressions to see whether the message should be sent to the recipient directly, or redirected to another peer. For instance, a rule stating that targets matching the expression $.*@example.com$ should be redirected to $gateway@p2p.example.com$, would result in all signaling to users of the $example.com$ domain to be routed to the peer $gateway@p2p.example.com$.

A remote gateway peer consists of two functionally separate entities: the gateway peer (a P2P peer), and the attached gateway adaptor. The gateway peer receives the redirected signaling and compares it against its own set of rules. These rules includes who is allowed to use the gateway (the source peer of the SIP signaling), for whom it relays (the recipient of the messages) and where the gateway adaptor is located (network address). If the signaling conforms to these rules, the SIP message is sent to the gateway adaptor for processing. The gateway adaptor is also responsible for translating the identities of the SIP signaling into those used by the external system through an account-mapping database.

The gateway adaptor is in this way responsible for maintaining the user accounts of the external systems and mapping these to the P2P identities. For instance, a gateway to Skype might map the identity $alice@p2psip.info$ to $alice\_on\_skype$. A one-to-one mapping is not strictly necessary (all identities can be configured to a single one), but this would lead to all calls from within the P2P system seeming to be coming from the same user. Furthermore, receiving calls from the external system would not be possible, as the external identity maps to several P2P identities. The gateway adaptor performs this substitution and forwards the message to the proxy of the external system. Figure 9.1 illustrates this flow. The gateway adaptor is naturally also responsible for keeping the registrations to the external systems alive. This might cause confusion when placing calls from the external system, as the user may seem available (as the registration is maintained by the gateway adaptor), but the actual recipient is not.

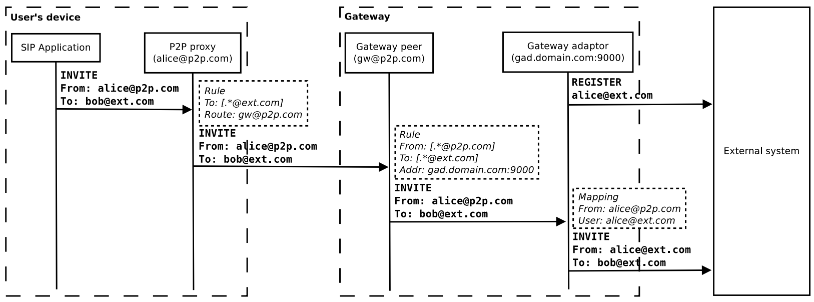

User's device
SIP Application
P2P proxy (alice@p2p.com)
Gateway
Gateway peer (gw@p2p.com)
Gateway adaptor (gad.domain.com:9000)
External system
INVITE From: alice@p2p.com To: bob@ext.com
Rule To: [.*@ext.com] Route: gw@p2p.com
INVITE From: alice@p2p.com To: bob@ext.com
Rule From: [.*@p2p.com] To: [.*@ext.com] Addr: gad.domain.com:9000
INVITE From: alice@p2p.com To: bob@ext.com
REGISTER alice@ext.com
Mapping From: alice@p2p.com User: alice@ext.com
INVITE From: alice@ext.com To: bob@ext.com

Figure 9.1. Sample signaling flow when placing a call to a user of an external system.

The gateway adaptor can be part of the local proxy, but is designed as a separate entity and may be moved to another location, as discussed. In general, as it will contain the usernames and passwords to the external system for all the users it serves, it will be a trusted authority and protected accordingly. This does create a lucrative target for attacks, which is a weakness of the design. However, as noted, some systems may require such a set-up, as the interface to the external system might not be accessible to normal users, such as company-internal telephone systems. Systems with weak security can also benefit from being accessible only through a gateway peer, as they can leverage the strong security offered by the P2P framework. But for accessing normal public systems, a dedicated remote peer is not needed.

9.2 Future work

The framework presented in this dissertation has been designed as a platform for exploring different issues applications operating in future networks may face. We have concentrated our efforts on building a generic model deployable in current networks, compatible with existing applications. The model is simple, providing applications with basic connectivity and identity management, designed to be portable to different types of network environments. This has provided a good start, allowing us to rapidly develop our system and concentrate on addressing security issues. To proceed however, we will need to look beyond integrating existing applications and consider how future applications, aware of the network environment, are designed.

We will continue the work by designing a run-time environment for *native* applications of the framework. This environment will provide applications greater access to the overlay network, not only for connection establishment, but also for features such as shared storage. We believe that this will allow us to examine how future applications will use network services, which correlates to how the networks need to be structured. Our aim is to develop an interface which provides applications with a sufficient amount of flexibility to accomplish complex tasks while remaining simple and adaptable to different environments.

This will take us towards more data-oriented networking architectures and publish-subscribe paradigms, which we have so far only brushed upon, to be able to achieve the level of efficiency needed. Although many such models have been proposed, the details of *how* exactly different types of applications would be implemented in these environments has not been explored. We would examine the most popular networking applications and services used today and see how, if at all, they can be adapted to distributed environments. In these environments, we imagine each peer having the applications themselves, but often an incomplete or not current view of the data set. This raises questions about who is allowed to modify the shared database, or whose input we accept, which can result in highly subjective views of the available data. This is what we intent to explore with the native applications.

Another development track we will continue on is the identity management. As the legacy applications are currently unaware of the cross-application identities, we are unable to fully leverage the potential synergy. By allowing applications access to the identity management and trust database could enable more precise and secure application behavior.

10. Conclusions

As the Internet grows closer to the limits of what the current architecture can handle, we look for new solutions in network management and application design which would allow for more robust, scalable and efficient networking. Currently network services are dominated by centralized models. Large data sets are stored on (logically) single nodes, and users are forced to route traffic great distances through remote servers, even when both parties are physically near, sharing the same local access point.

Distributed models and use of P2P technologies in application design has been a natural step towards a more scalable networking environment. On the network level, we see data-oriented and publish-subscribe architectures proposed as solutions, providing better network efficiency for many applications.

The concept of P2P and distributed applications is well known and there exists a myriad of different implementations today. However, these are exclusively highly focused applications, completing only a single task or providing a specific service. For each one, new infrastructure has to be deployed, security issues resolved and an identity scheme established. As we look for new solutions, we must also realize that it is not only the size of the network, the amount of content and users, that has grown, but the way the network is used has changed as well. Today we access the Internet from a variety of different devices in different places, using a multitude of access technologies.

In this dissertation, we have explored the implications of a generic, cross-application, P2P framework. With this framework, we have tried to simulate a future networking environment that applications may face, and studied security and other issues associated with it. We have shown through a prototype implementation that such a framework can be deployed in existing networks by using existing infrastructure and advanced networking protocols, and can be used even by current, existing, applications.

Much of our research has revolved around identity management and trust. We have identified, and addressed, privacy issues and experimented with leveraging cross-application identities to create a safer networking environment. Finally, we have highlighted the importance of identity mobility and how these may form the cornerstone of networking security in the future.

We feel that our work has been fruitful, and provided a way of experimenting with different models for how applications could work in the networking environments we are headed for. Much of the research on future networking has been concentrated on either efficient network architectures or enhancing the application use in the current architecture. In this dissertation, we have combined these and examined how applications could be enhanced and adopted to a future networking environment.

Bibliography

[1] Mailvelope–openpgp encryption for webmail. https://www.mailvelope.com/.

[2] The <module> tag. http://json.org/module.html.

[3] Openid foundation website. http://openid.net/. Referenced: 2010-02-24.

[4] Webpg–bringing gnupg/pgp to the web browser. https://webpg.org/.

[5] Karl Aberer and Zoran Despotovic. Managing trust in a peer-2-peer information system. In *Proceedings of the tenth international conference on Information and knowledge management*, CIKM '01, pages 310–317, New York, NY, USA, 2001. ACM.

[6] Mark S. Ackerman, Lorrie Faith Cranor, and Joseph Reagle. Privacy in e-commerce: examining user scenarios and privacy preferences. In *Proceedings of the ACM Conference on Electronic Commerce*, 1999.

[7] Anne Adams and Martina Angela Sasse. Users are not the enemy. *Commun. ACM*, 42(12):40–46, December 1999.

[8] Eytan Adar and Bernardo A. Huberman. Free riding on Gnutella. *First Monday*, 5(10), October 2000.

[9] Daniel Adkins, Karthik Lakshminarayanan, Adrian Perrig, and Ion Stoica. Towards a more functional and secure network infrastructure. Technical Report UCB/CSD-03-1242, 2003.

[10] Mark Allman, Christian Kreibich, Vern Paxson, Robin Sommer, and Nicholas Weaver. The strengths of weaker identities: opportunistic personas. In *Proceedings of the 2nd USENIX workshop on Hot topics in security (HOTSEC'07)*, pages 1–6, Berkeley, CA, USA, 2007. USENIX Association.

[11] Paulo Sérgio Almeida, Carlos Baquero, Nuno Preguiça, and David Hutchison. Scalable bloom filters. *Inf. Process. Lett.*, 101(6):255–261, March 2007.

[12] Artur Andrzejak and Zhichen Xu. Scalable, efficient range queries for grid information services. In *Proceedings of the Second International Conference on Peer-to-Peer Computing (P2P '02)*, page 33, Washington, DC, USA, 2002. IEEE Computer Society.

[13] Apache HTTP Server Project. http://httpd.apache.org/.

[14] Niranjan Balasubramanian, Aruna Balasubramanian, and Arun Venkataramani. Energy consumption in mobile phones: a measurement study and implications for network applications. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, IMC '09, pages 280–293, New York, NY, USA, 2009. ACM.

[15] Dirk Balfanz, Glenn Durfee, Rebecca E. Grinter, and Diana K. Smetters. In search of usable security: Five lessons from the field. *IEEE Security & Privacy*, 2(5):19–24, 2004.

[16] Farnoush Banaei-kashani, Ching chien Chen, and Cyrus Shahabi. Wspds: Web services peer-to-peer discovery service. In *Proceedings of the International Conference on Internet Computing*, pages 733–743, 2004.

[17] S. A. Baset and H. G. Schulzrinne. An analysis of the skype Peer-to-Peer internet telephony protocol. In *Proceedings 25th IEEE International Conference on Computer Communications (INFOCOM 2006)*, 2006.

[18] M. Baugher, D. McGrew, M. Naslund, E. Carrara, and K. Norrman. The Secure Real-time Transport Protocol (SRTP). RFC 3711 (Proposed Standard), March 2004. Updated by RFC 5506.

[19] Ingmar Baumgart. P2pns: A secure distributed name service for p2psip. In *Proceedings of the 2008 Sixth Annual IEEE International Conference on Pervasive Computing and Communications*, pages 480–485, Washington, DC, USA, 2008. IEEE Computer Society.

[20] A. R. Bharambe, C. Herley, and V. N. Padmanabhan. Analyzing and Improving a BitTorrent Networks Performance Mechanisms. In *Proceedings of the 25th IEEE International Conference on Computer Communications (INFOCOM 2006)*, April 2006.

[21] Ashwin R. Bharambe, Mukesh Agrawal, and Srinivasan Seshan. Mercury: supporting scalable multi-attribute range queries. *SIGCOMM Comput. Commun. Rev.*, 34(4):353–366, 2004.

[22] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13:422–426, July 1970.

[23] David A. Bryan, Bruce Lowekamp, and Cullen Jennings. Sosimple: A serverless, standards-based, p2p sip communication system. In *AAA-IDEA*, pages 42–49. IEEE Computer Society, 2005.

[24] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer. OpenPGP Message Format. RFC 4880 (Proposed Standard), November 2007. Updated by RFC 5581.

[25] G. Camarillo and A. Keränen. Host Identity Protocol (HIP) Multi-Hop Routing Extension. RFC 6028 (Experimental), October 2010.

[26] G. Camarillo and J. Melen. HIP (Host Identity Protocol) Immediate Carriage and Conveyance of Upper- layer Protocol Signaling (HICCUPS). http://www.ietf.org/internet-drafts/draft-ietf-hip-hiccups-05.txt, July 2010. Work in progress.

[27] G. Camarillo, P. Nikander, and J. Hautakorpi. IP BONE: Host Identity Protocol (HIP) Based Overlay Networking Environment. http://www.ietf.org/internet-drafts/draft-camarillo-hip-bone-00.txt, November 2007. Work in progress.

[28] B. Campbell, R. Mahy, and C. Jennings. The Message Session Relay Protocol (MSRP). RFC 4975 (Proposed Standard), September 2007.

[29] B. Campbell, J. Rosenberg, H. Schulzrinne, C. Huitema, and D. Gurle. Session Initiation Protocol (SIP) Extension for Instant Messaging. RFC 3428 (Proposed Standard), December 2002.

[30] B. Carpenter and K. Moore. Connection of IPv6 Domains via IPv4 Clouds. RFC 3056 (Proposed Standard), February 2001.

[31] Maciej Ceglowski and Joshua Schachter. Loaf, a simple emailextension for contactrecognition using bloomfilter. http://loaf.cantbedone.org. Referenced: 2010-02-24.

[32] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Proceedings of the International Workshop on Designing Privacy Enhancing Technologies: Design Issues in Anonymity and Unobservability*, pages 46–66, New York, NY, USA, 2001. Springer-Verlag New York, Inc.

[33] Florence Clevenot and Philippe Nain. A simple fluid model for the analysis of the squirrel peer-to-peer caching system. In *Proceedings of IEEE INFOCOM*, Hong Kong, March 2004.

[34] Bram Cohen. Incentives build robustness in BitTorrent. http://bitconjurer.org/BitTorrent/bittorrentecon.pdf, May 2003.

[35] N.J. Croft and M.S. Olivier. A model for spam prevention in IP telephony networks using anonymous verifying authorities. In *Proceedings of the ISSA New Knowledge Today Conference*, April 2005.

[36] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with cfs. *SIGOPS Oper. Syst. Rev.*, 35:202–215, October 2001.

[37] Ernesto Damiani, De Capitani di Vimercati, Stefano Paraboschi, Pierangela Samarati, and Fabio Violante. A reputation-based approach for choosing reliable resources in peer-to-peer networks. In *Proceedings of the 9th ACM conference on Computer and communications security*, CCS '02, pages 207–216, New York, NY, USA, 2002. ACM.

[38] Alexander J. DeWitt and Jasna Kuljis. Aligning usability and security: a usability study of polaris. In Lorrie Faith Cranor, editor, *SOUPS*, volume 149 of *ACM International Conference Proceeding Series*, pages 1–7. ACM, 2006.

[39] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008.

[40] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[41] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, pages 21–21, Berkeley, CA, USA, 2004. USENIX Association.

[42] John R. Douceur. The sybil attack. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems (IPTPS '01)*, pages 251–260, London, UK, 2002. Springer-Verlag.

[43] D. Eastlake 3rd and P. Jones. US Secure Hash Algorithm 1 (SHA1). RFC 3174 (Informational), September 2001. Updated by RFC 4634.

[44] Fatih Emekci, Ozgur D. Sahin, Divyakant Agrawal, and Amr El Abbadi. A peer-to-peer framework for web service discovery with ranking. In *Proceedings of the IEEE International Conference on Web Services (ICWS '04)*, page 192, Washington, DC, USA, 2004. IEEE Computer Society.

[45] Eyeball AntiSPIT Server. http://www.eyeball.com/products/anti_spit_server.html.

[46] Michael Farb, Yue-Hsun Lin, Tiffany Hyun-Jin Kim, Jonathan McCune, and Adrian Perrig. Safeslinger: easy-to-use and secure public-key exchange. In *Proceedings of the 19th annual international conference on Mobile computing and networking*, MobiCom '13, pages 417–428, New York, NY, USA, 2013. ACM.

[47] Michal Feldman and John Chuang. Overcoming free-riding behavior in peer-to-peer systems. *SIGecom Exch.*, 5:41–50, July 2005.

[48] Michal Feldman, Christos Papadimitriou, John Chuang, and Ion Stoica. Free-riding and whitewashing in peer-to-peer systems. In *Proceedings of the ACM SIGCOMM workshop on Practice and theory of incentives in networked systems*, PINS '04, pages 228–236, New York, NY, USA, 2004. ACM.

[49] Bryan Ford, Pyda Srisuresh, and Dan Kegel. Peer-to-peer communication across network address translators. In *Proceedings of the annual conference on USENIX Annual Technical Conference (ATEC '05)*, pages 13–13, Berkeley, CA, USA, 2005. USENIX Association.

[50] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart. HTTP Authentication: Basic and Digest Access Authentication. RFC 2617 (Draft Standard), June 1999.

[51] Prasanna Ganesan, Beverly Yang, and Hector Garcia-molina. One torus to rule them all: Multi-dimensional queries in p2p systems. In *Proceedings of the 7th International Workshop on the Web and Databases (WebDB'04)*. ACM Press, 2004.

[52] Guido Gehlen and Linh Pham. Mobile web services for peer-to-peer applications. In *Consumer Communications and Networking Conference, 2005. CCNC. 2005 Second IEEE*, pages 427 – 433, 2005.

[53] Dimitris Geneiatakis, Georgios Kambourakis, Tasos Dagiuklas, Costas Lambrinoudakis, and Stefanos Gritzalis. SIP security mechanisms: A state-of-the-art review. In *Proceedings of the 5th International Network Conference*, pages 147–155. ACM, 2005.

[54] Nathaniel S. Good and Aaron Krekelberg. Usability and privacy: a study of kazaa p2p file-sharing. In *Proceedings of the SIGCHI conference on Human factors in computing systems (CHI '03)*, pages 137–144, New York, NY, USA, 2003. ACM.

[55] Ben Greenstein, Damon McCoy, Jeffrey Pang, Tadayoshi Kohno, Srinivasan Seshan, and David Wetherall. Improving wireless privacy with an identifier-free link layer protocol. In *Proceeding of the 6th international conference on Mobile systems, applications, and services (MobiSys '08)*, pages 40–53, New York, NY, USA, 2008. ACM.

[56] Saikat Guha, Neil Daswani, and Ravi Jain. An experimental study of the skype peer-to-peer voip system. In Emin Gün Sirer and Ben Y. Zhao, editors, *IPTPS*, 2006.

[57] Deke Guo, Honghui Chen, and Xueshan Luo. Theory and network applications of dynamic bloom filters. In *Proceedings of the 25th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM*, 2006.

[58] Abhishek Gupta, Divyakant Agrawal, and Amr El Abbadi. Approximate range selection queries in peer-to-peer systems. In *Proceedings of the First Biennal Conference on Innovative Data Systems Research (CIDR)*, pages 141–151, 2003.

[59] Andrei Gurtov. *Host Identity Protocol (HIP): Towards the Secure Mobile Internet*. Wiley and Sons, 2008.

[60] Selim Gurun, Priya Nagpurkar, and Ben Y. Zhao. Energy consumption and conservation in mobile peer-to-peer systems. In *Proceedings of the 1st international workshop on Decentralized resource sharing in mobile computing and networking*, MobiShare '06, pages 18–23, New York, NY, USA, 2006. ACM.

[61] E. Guttman, C. Perkins, J. Veizades, and M. Day. Service Location Protocol, Version 2. RFC 2608 (Proposed Standard), June 1999. Updated by RFC 3224.

[62] Matthew Harren, Joseph M. Hellerstein, Ryan Huebsch, Boon Thau Loo, Scott Shenker, and Ion Stoica. Complex queries in dht-based peer-to-peer networks. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems (IPTPS '01)*, pages 242–259, London, UK, 2002. Springer-Verlag.

[63] G. Lopez D. Hautakorpi, J. Camarillo. Framework for decentralizing legacy applications. In *Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID '09)*, pages 544 – 549.

[64] J. Hautakorpi, G. Camarillo, and J. Koskela. Utilizing HIP (Host Identity Protocol) for P2PSIP (Peer-to-peer Session Initiation Protocol). http://www.ietf.org/internet-drafts/draft-camarillo-hip-bone-00.txt, December 2007. Work in progress.

[65] J. Hautakorpi, A. Salinas, E. Harjula, and M. Ylianttila. Interconnecting P2PSIP and IMS. In *Proceedings of the Second International Conference on*

Next Generation Mobile Applications, Services and Technologies (NGMAST '08), pages 83 – 88, Sep 2008.

[66] Juho Heikkilä and Andrei Gurtov. Filtering SPAM in P2PSIP communities with Web of Trust. In *Proceedings of the First International ICST Conference on Security and Privacy in Mobile Information and Communication Systems (MobiSec 2009)*, pages 110–121, 2009.

[67] HIP for inter.net Project. http://www.hip4inter.net/.

[68] HIP for Linux. http://hipl.hiit.fi.

[69] Liao Hongluan, Peng Jin, Yu Zhenyu, and Wang Yiwen. A reload usage for distributed conference media processing. http://www.ietf.org/internet-drafts/draft-liao-p2psip-dcmp-01.txt, October 2010. Work in progress.

[70] Shiang-Ming Huang, Quincy Wu, and Yi-Bing Lin. Enhancing teredo ipv6 tunneling to traverse the symmetric nat. *Communications Letters, IEEE*, 10:408 – 410, May 2006.

[71] C. Huitema. Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs). RFC 4380 (Proposed Standard), February 2006.

[72] IETF HIP working group. http://www.ietf.org/html.charters/hip-charter.html.

[73] IETF P2PSIP WG. http://www.ietf.org/dyn/wg/charter/simple-charter.html.

[74] IETF P2PSIP working group. http://www.ietf.org/html.charters/p2psip-charter.html.

[75] Tomas Isdal, Michael Piatek, Arvind Krishnamurthy, and Thomas Anderson. Privacy-preserving p2p data sharing with oneswarm. In *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, pages 111–122, New York, NY, USA, 2010. ACM.

[76] Sitaram Iyer, Antony Rowstron, and Peter Druschel. Squirrel: a decentralized peer-to-peer web cache. In *Proceedings of the twenty-first annual symposium on Principles of distributed computing (PODC '02)*, pages 213–222, New York, NY, USA, 2002. ACM.

[77] M. Izal, Guillaume Urvoy-Keller, Ernst W. Biersack, P. A. Felber, A. Al Hamra, and L. Garcés-Erice. Dissecting BitTorrent: Five Months in a Torrent's Lifetime. In *Passive and Active Network Measurement*, pages 1–11. 2004.

[78] C. Jennings, B. Lowekamp, E. Rescorla, S. Baset, and H. Schulzrinne. REsource LOcation And Discovery (RELOAD). http://www.ietf.org/internet-drafts/draft-ietf-p2psip-reload-00.txt, July 2008. Work in progress.

[79] C. Jennings, B. Lowekamp, E. Rescorla, S. Baset, and H. Schulzrinne. REsource LOcation And Discovery (RELOAD) Base Protocol. http://www.ietf.org/internet-drafts/draft-ietf-p2psip-base-07.txt, February 2010. Work in progress.

[80] C. Jennings, B. Lowekamp, E. Rescorla, S. Baset, and H. Schulzrinne. A sip usage for reload. http://www.ietf.org/internet-drafts/draft-ietf-p2psip-sip-05.txt, July 2010. Work in progress.

[81] C. Jennings, J. Peterson, and M. Watson. Private Extensions to the Session Initiation Protocol (SIP) for Asserted Identity within Trusted Networks. RFC 3325 (Informational), November 2002.

[82] P. Jokela, R. Moskowitz, and P. Nikander. Using ESP transport format with HIP. http://www.ietf.org/internet-drafts/draft-ietf-hip-esp-06.txt, June 2007. Work in progress.

[83] K. Singh and H. Schulzrinne. Peer-to-peer internet telephony using sip. In Wu chi Feng and Ketan Mayer-Patel, editors, *NOSSDAV*, pages 63–68. ACM, 2005.

[84] Linda Källström, Simone Leggio, Jukka Manner, Tommi Mikkonen, Kimmo E. E. Raatikainen, Jussi Saarinen, Sanna Suoranta, and Antti Ylä-Jääski. A framework for seamless service interworking in ad-hoc networks. *Computer Communications*, 29(16):3277–3294, 2006.

[85] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *Proceedings of the 12th international conference on World Wide Web*, WWW '03, pages 640–651, New York, NY, USA, 2003. ACM.

[86] Chris Kanich, Christian Kreibich, Kirill Levchenko, Brandon Enright, Geoffrey M. Voelker, Vern Paxson, and Stefan Savage. Spamalytics: an empirical analysis of spam marketing conversion. In *Proceedings of the 15th ACM conference on Computer and communications security*, CCS '08, pages 3–14, New York, NY, USA, 2008. ACM.

[87] Murat Karakaya, Ibrahim Korpeoglu, and Özgür Ulusoy. Free riding in peer-to-peer networks. *IEEE Internet Computing*, 13:92–98, March 2009.

[88] Otso Kassinen, Erkki Harjula, and Mika Ylianttila. Suitability of dht-based peer-to-peer session initiation protocol for wireless distributed services. In *Proceedings 12th International Symposium on Wireless Personal Multimedia Communications*, September 2009.

[89] Otso Kassinen, Zhonghong Ou, Mika Ylianttila, and Erkki Harjula. Effects of peer-to-peer overlay parameters on mobile battery duration and resource lookup efficiency. In *Proceedings of the 7th International Conference on Mobile and Ubiquitous Multimedia*, MUM '08, pages 177–180, New York, NY, USA, 2008. ACM.

[90] Koskela T Kassinen O and Ylianttila M. Using unstructured service supplementary data signaling for mobile peer-to-peer invocations. In *Proceedings 12th International Symposium on Wireless Personal Multimedia Communications*, Sendai, Japan, September 2009.

[91] I. Kelenyi and J.K. Nurminen. Energy-consumption in mobile peer-to-peer - quantitative results from file sharing. In *IEEE International Conference on Communications Workshops, 2008. ICC Workshops '08.*, pages 164 – 168, Beijing, May 2008.

[92] Imre Kelényi and Jukka K. Nurminen. Optimizing energy consumption of mobile nodes in heterogeneous kademlia-based distributed hash tables. In *Proceedings of the 2008 The Second International Conference on Next Generation Mobile Applications, Services, and Technologies*, pages 70–75, Washington, DC, USA, 2008. IEEE Computer Society.

[93] A. Keränen, G. Camarillo, and J. Mäenpaa. Host Identity Protocol-Based Overlay Networking Environment (HIP BONE) Instance Specification for REsource LOcation And Discovery (RELOAD). http://www.ietf.org/internet-drafts/draft-ietf-hip-reload-instance-03 .txt, January 2011. Work in progress.

[94] A. Khurri, E. Vorobyeva, and A. Gurtov. Performance of host identity protocol on lightweight hardware. In *Proceedings of the ACM SIGCOMM Workshop on Mobility in the Evolving Internet Architcture MobiArch'07*, pages 27–34. ACM, 2007.

[95] A. Knauf, G. Hege, T C. Schmidt, and M. Waehlisch. A reload usage for distributed conference control (DisCo). http://www.ietf.org/internet-drafts/draft-knauf-p2psip-disco-01.txt, December 2010. Work in progress.

[96] Teemu Koponen, Mohit Chawla, Byung-Gon Chun, Andrey Ermolinskiy, Kye Hyun Kim, Scott Shenker, and Ion Stoica. A data-oriented (and beyond) network architecture. *SIGCOMM Comput. Commun. Rev.*, 37(4):181–192, 2007.

[97] Dmitry Korzun, Boris Nechaev, and Andrei Gurtov. Cyclic routing: Generalizing lookahead in peer-to-peer networks. In *Proceedings ot the 7th IEEE/ACS Int'l Conf. on Computer Systems and Applications (AICCSA2009)*, pages 697–704. IEEE Computer Society, May 2009.

[98] Joakim Koskela. A HIP-based peer-to-peer communication system. In *Proceedings of the 15th International Conference on Telecommunications (ICT2008)*, pages 1–7, June 2008.

[99] Joakim Koskela and Andrei Gurtov. A secure peer-to-peer web framework. In *Proceedings of the 2010 IEEE International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, WOWMOM '10, pages 1–6, Washington, DC, USA, 2010. IEEE Computer Society.

[100] Joakim Koskela, Juho Heikkilä, and Andrei Gurtov. Poster abstract: a secure P2P SIP system with SPAM prevention. *Mobile Computing and Communications Review*, (3):26–29.

[101] Joakim Koskela, Kristiina Karvonen, and Theofanis Kilinkaridis. Usable and secure P2P VoIP for mobile use. In *International Journal of Mobile Human Computer Interaction (IJMHCI) Accepted for publication*, Hershey, PA, USA, 2015. IGI Global.

[102] Joakim Koskela, Kristiina Karvonen, Theofanis Kilinkaridis, and Andrei Gurtov. Secure and usable P2P VoIP for mobile devices. In *Proceedings of the 12th international conference on Human computer interaction with mobile devices and services*, MobileHCI '10, pages 439–442, New York, NY, USA, 2010. ACM.

[103] Joakim Koskela and Sasu Tarkoma. Simple peer-to-peer SIP privacy. In *Proceedings of the First International ICST Conference on Security and Privacy in Mobile Information and Communication Systems (MobiSec 2009)*, pages 226–237, 2009.

[104] Joakim Koskela, Nicholas Weaver, Andrei Gurtov, and Mark Allman. Securing web content. In *Proceedings of the 2009 workshop on Re-architecting the internet (ReArch '09)*, pages 7–12, New York, NY, USA, 2009. ACM.

[105] Timo Koskela, Janne Julkunen, Jari Korhonen, Meirong Liu, and Mika Ylianttila. Leveraging collaboration of peer-to-peer and web services. In *Proceedings of the 2008 The Second International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies*, pages 496–501, Washington, DC, USA, 2008. IEEE Computer Society.

[106] Timo Koskela, Nonna Kostamo, Otso Kassinen, Juuso Ohtonen, and Mika Ylianttila. Towards context-aware mobile web 2.0 service architecture. In *Proceedings of the International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies*, pages 41–48, Washington, DC, USA, 2007. IEEE Computer Society.

[107] TaiJin Lee, JongIl Jeong, HyungJong Kim, HyunChul Jeong, and YooJae Won. User reputation based voip spam defense architecture. In *Proceedings of the 23rd international conference on Information Networking (ICOIN'09)*, pages 161–165, Piscataway, NJ, USA, 2009. IEEE Press.

[108] Simone Leggio, Jukka Manner, and Kimmo E. E. Raatikainen. A secure sip-based instant messaging and presence framework for ad-hoc networks. In *GLOBECOM*. IEEE, 2006.

[109] Dave Levin, Katrina LaCurts, Neil Spring, and Bobby Bhattacharjee. Bittorrent is an auction: analyzing and improving bittorrent's incentives. In *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, SIGCOMM '08, pages 243–254, New York, NY, USA, 2008. ACM.

[110] Prakash Linga, Indranil Gupta, and Ken Birman. A churn-resistant peer-to-peer web caching system. In *Proceedings of the 2003 ACM workshop on Survivable and self-regenerative systems (SSRS '03)*, pages 1–10, New York, NY, USA, 2003. ACM.

[111] Yi Lu, Weichao Wang, B. Bhargava, and Dongyan Xu. Trust-based privacy preservation for peer-to-peer data sharing. *Systems, Man and Cybernetics, Part A, IEEE Transactions on*, 36(3):498–502, May 2006.

[112] Qin Lv, Pei Cao, Edith Cohen, Kai Li, and Scott Shenker. Search and replication in unstructured peer-to-peer networks. In *Proceedings of the 16th international conference on Supercomputing*, ICS '02, pages 84–95, New York, NY, USA, 2002. ACM.

[113] D. MacIntosh, R. Vinokurov. Detection and mitigation of spam in ip telephony networks using signaling protocol analysis. In *Advances in Wired and Wireless Communication, 2005 IEEE/Sarnoff Symposium on*.

[114] Verdi March and Yong Meng Teo. Multi-attribute range queries on read-only dht. In *Computer Communications and Networks, 2006. ICCCN 2006.*

Proceedings.15th International Conference on, pages 419–424, Arlington, VA, USA, 2006.

[115] M Matuszewski and E Kokkonen. Mobile p2psip - peer-to-peer sip communication in mobile communities. In *Consumer Communications and Networking Conference, 2008. CCNC 2008. 5th IEEE*, pages 1159 – 1165, Las Vegas, NV, Jan 2008.

[116] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, pages 53–65, London, UK, 2002. Springer-Verlag.

[117] Messaging Anti-Abuse Working Group. Email metrics program: the network operator's perspective, report #7, April 2008.

[118] Miredo : Teredo IPv6 tunneling for Linux and BSD. http://www.remlab.net/miredo/.

[119] Mobile Web Server. http://betalabs.nokia.com/betas/view/mobile-web-server.

[120] Anirban Mondal and Masaru Kitsuregawa. Privacy, security and trust in p2p environments: A perspective. *International Workshop on Database and Expert Systems Applications*, pages 682–686, 2006.

[121] R. Moskowitz and P. Nikander. Host Identity Protocol (HIP) Architecture. RFC 4423 (Informational), May 2006.

[122] P. Nikander, J. Arkko, and B. Ohlman. Host identity indirection infrastructure (hi3). In *The Second Swedish National Computer Networking Workshop*, November 2004.

[123] P. Nikander, J. Laganier, and F. Dupont. An IPv6 Prefix for Overlay Routable Cryptographic Hash Identifiers (ORCHID). RFC 4843 (Experimental), April 2007.

[124] Donald A. Norman. The way I see it: When security gets in the way. *ACM interactions*, 16(6):60–63, November 2009.

[125] J.K. Nurminen and J. Noyranen. Energy-consumption in mobile peer-to-peer - quantitative results from file sharing. In *Consumer Communications and Networking Conference, 2008. CCNC 2008. 5th IEEE*, pages 729 – 733, Las Vegas, NV, Jan 2008.

[126] J. Korhonen M. Ylianttila O. Kassinen, E. Harjula. Battery life of mobile peers with umts and wlan in a kademlia-based p2p overlay. In *Proceedings of the 20th Personal, Indoor and Mobile Radio Communications Symposium (PIMRC09)*, Tokyo, Japan, Sep 2009.

[127] OpenDHT: A Publicly Accessible DHT Service. http://www.opendht.org/.

[128] Z. Ou, E. Harjula, and M. Ylianttila. Effects of different churn models on the performance of structured peer-to-peer networks. In *Proceedings of the 20th Personal, Indoor and Mobile Radio Communications Symposium (PIMRC09)*, Tokyo, Japan, Sep 2009.

[129] Zhonghong Ou, Erkki Harjula, Otso Kassinen, and Mika Ylianttila. Feasibility evaluation of a communication-oriented p2p system in mobile environments. In *Mobility Conference*. ACM, 2009.

[130] Zhonghong Ou, Erkki Harjula, Otso Kassinen, and Mika Ylianttila. Performance evaluation of a kademlia-based communication-oriented p2p system under churn. *Comput. Netw.*, 54:689–705, April 2010.

[131] Y. Peng and Y. Meng. An snmp usage for reload. http://www.ietf.org/internet-drafts/draft-peng-p2psip-snmp-00.txt, October 2010. Work in progress.

[132] J. Peterson and C. Jennings. Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP). RFC 4474 (Proposed Standard), August 2006.

[133] Michael Piatek, Tomas Isdal, Thomas Anderson, Arvind Krishnamurthy, and Arun Venkataramani. Do incentives build robustness in BitTorrent? In *Proceedings of 4th USENIX Symposium on Networked Systems Design & Implementation (NSDI 2007)*, Cambridge, MA, April 2007. USENIX.

[134] PlanetLab: An open platform for developing, deploying and accessing planetary-scale services. http://www.planet-lab.org/.

[135] Johan Pouwelse, Pawel Garbacki, Dick Epema, and Henk Sips. The Bittorrent P2P File-Sharing System: Measurements and Analysis. In Miguel Castro and Robbert van Renesse, editors, *Peer-to-Peer Systems IV*, volume 3640 of *Lecture Notes in Computer Science*, chapter 19, pages 205–216. Springer Berlin / Heidelberg, Berlin/Heidelberg, 2005.

[136] Dongyu Qiu and R. Srikant. Modeling and performance analysis of bittorrent-like peer-to-peer networks. *SIGCOMM Comput. Commun. Rev.*, 34:367–378, August 2004.

[137] Jürgen Quittek, Saverio Niccolini, Sandra Tartarelli, Martin Stiemerling, Marcus Brunner, and Thilo Ewald. Detecting spit calls by checking human communication patterns. In *ICC*, pages 1979–1984. IEEE, 2007.

[138] B. Ramsdell. Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification. RFC 3851 (Proposed Standard), July 2004.

[139] Sylvia Ratnasamy, Paul Francis, Scott Shenker, and Mark Handley. A scalable content-addressable network. In *Proceedings of ACM SIGCOMM*, pages 161–172, 2001.

[140] Thomas Repantis and Vana Kalogeraki. Decentralized trust management for ad-hoc peer-to-peer networks. In *Proceedings of the 4th international workshop on Middleware for Pervasive and Ad-Hoc Computing (MPAC 2006)*, MPAC '06, pages 6–, New York, NY, USA, 2006. ACM.

[141] E. Rescorla. HTTP Over TLS. RFC 2818 (Informational), May 2000. Updated by RFC 5785.

[142] R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21:120–126, 1978.

[143] J. Rosenberg. A Presence Event Package for the Session Initiation Protocol (SIP). RFC 3856 (Proposed Standard), August 2004.

[144] J. Rosenberg. Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols. http://www.ietf.org/internet-drafts/draft-ietf-mmusic-ice-19.txt, October 2007. Work in progress.

[145] J. Rosenberg and C. Jennings. The Session Initiation Protocol (SIP) and Spam. RFC 5039 (Informational), January 2008.

[146] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261 (Proposed Standard), June 2002. Updated by RFCs 3265, 3853, 4320, 4916.

[147] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, November 2001.

[148] Antony Rowstron, Anne-Marie Kermarrec, Miguel Castro, and Peter Druschel. Scribe: The design of a large-scale event notification infrastructure. *Networked Group Communication*, pages 30–43, 2001.

[149] Martina Angela Sasse and Ivan Flechais. Usable security – why do we need it? how do we get it? *Security and Usability: Designing secure systems that people can use*, page 18, 2005.

[150] Stuart E. Schechter, Rachna Dhamija, Andy Ozment, and Ian Fischer. The emperor's new security indicators. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, SP '07, pages 51–65, Washington, DC, USA, 2007. IEEE Computer Society.

[151] Stuart E. Schechter, Rachna Dhamija, Andy Ozment, and Ian Fischer. Emperors new security indicators: An evaluation of website authentication and the effect of role playing on usability studies. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, 2007.

[152] Cristina Schmidt and Manish Parashar. A peer-to-peer approach to web service discovery. *World Wide Web*, 7(2):211–229, 2004.

[153] Huaping Shen, Mary Suchitha Joseph, Mohan Kumar, and Sajal K. Das. Precinct: A scheme for cooperative caching in mobile peer-to-peer systems. *International Parallel and Distributed Processing Symposium*, 1:57a, 2005.

[154] Alex Sherman, Jason Nieh, and Clifford Stein. Fairtorrent: bringing fairness to peer-to-peer systems. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, CoNEXT '09, pages 133–144, New York, NY, USA, 2009. ACM.

[155] Rob Sherwood, Ryan Braud, and Bobby Bhattacharjee. Slurpie: A cooperative bulk data transfer protocol. In *Proceedings of IEEE INFOCOM*, Hong Kong, March 2004.

[156] Shuming Shi, Guangwen Yang, Dingxing Wang, Jin Yu, Shaogang Qu, and Ming Chen. Making peer-to-peer keyword searching feasible using multi-level partitioning. In *Proceedings of the 3rd IPTPS*, pages 151–161, 2004.

[157] Weisong Shi, Kandarp Shah, Yonggen Mao, and Vipin Chaudhary. Tuxedo: A peer-to-peer caching system. In *Proceedings of PDPTA*, pages 981–987, 2003.

[158] Yanfeng Shu, Beng Chin Ooi, and Kian lee Tan. Supporting multi-dimensional range queries in peer-to-peer systems. In *Proceedings of the Fifth IEEE International Conference on Peer-to-Peer Computing (P2P 2005)*, pages 173–180, 2005.

[159] Jukka Manner Simone Leggio, Antti Hulkkonen, and Kimmo Raatikainen. Session initiation protocol deployment in ad-hoc networks: a decentralized approach. In *Proceedings of the International Workshop on Wireless Ad-Hoc Networks*, May 2005.

[160] A. Singh, T. W. Ngan, P. Druschel, and D. S. Wallach. Eclipse Attacks on Overlay Networks: Threats and Defenses. In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pages 1–12, 2006.

[161] K. Singh and H. Schulzrinne. SIP PEER: A session initiation protocol (sip)-based peer-to-peer internet telephony client adaptor. Ínternet, at http://www1.cs.columbia.edu/ kns10/publication/sip-p2p-design.pdf, 2005.

[162] Kundan Singh and Henning Schulzrinne. Using an external DHT as a SIP Location Service. Technical Report CUCS-007-06, Comp. Ci. Dept., Columbia University, 2006.

[163] spamhaus.org. Effective spam filtering. http://www.spamhaus.org/effective_filtering.html. Referenced: 2010-02-24.

[164] spamunit.com. Spam statistics. http://www.spamunit.com/spam-statistics/. Referenced: 2010-02-24.

[165] Tyron Stading, Petros Maniatis, and Mary Baker. Peer-to-peer caching schemes to address flash crowds. In *Peer-to-Peer Systems*, pages 203–213, Heidelberg, 2002. Springer Berlin.

[166] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet indirection infrastructure. In *Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM '02)*, pages 73–86, 2002.

[167] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '01, pages 149–160, New York, NY, USA, 2001. ACM.

[168] Jing Su, James Scott, Pan Hui, Jon Crowcroft, Eyal de Lara, Christophe Diot, Ashvin Goel, Menghow Lim, and Eben Upton. Haggle: Seamless networking for mobile applications. In John Krumm, Gregory D. Abowd,

Aruna Seneviratne, and Thomas Strang, editors, *Ubicomp*, volume 4717 of *Lecture Notes in Computer Science*, pages 391–408. Springer, 2007.

[169] Sasu Tarkoma, Christian Esteve Rothenberg, and Eemil Lagerspetz. Theory and practice of bloom filters for distributed systems. *IEEE Communications Surveys and Tutorials*, 14(1):131–155, 2012.

[170] The OpenHIP project. http://www.openhip.org/.

[171] Alan M. Turing. Computing Machinery and Intelligence. *Mind*, LIX:433–460, 1950.

[172] Andrew Twigg. A subjective approach to routing in p2p and ad hoc networks. In *Proceedings of the 1st international conference on Trust management*, iTrust'03, pages 225–238, Berlin, Heidelberg, 2003. Springer-Verlag.

[173] Johan Ugander, Brian Karrer, Lars Backstrom, and Cameron Marlow. The anatomy of the facebook social graph. *CoRR*, abs/1111.4503, 2011.

[174] Luis von Ahn, Manuel Blum, Nicholas J. Hopper, and John Langford. CAPTCHA: Using Hard AI Problems For Security. In *Proceedings of Eurocrypt*, volume 2656, pages 294–311, 2003.

[175] Limin Wang, Kyoung Soo Park, Ruoming Pang, Vivek Pai, and Larry Peterson. Reliability and security in the codeen content distribution network. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, ATEC '04, pages 14–14, Berkeley, CA, USA, 2004. USENIX Association.

[176] Alma Whitten and J. D. Tygar. Why johnny can't encrypt: A usability evaluation of pgp 5.0. In *Proceedings of the 8th Conference on USENIX Security Symposium - Volume 8*, SSYM'99, pages 14–14, Berkeley, CA, USA, 1999. USENIX Association.

[177] Wikipedia. Bloom filter. http://en.wikipedia.org/wiki/Bloom_filter. Referenced: 2010-02-24.

[178] Yu-Sung Wu, Saurabh Bagchi, Navjot Singh, and Ratsameetip Wita. Spam detection in voice-over-ip calls through semi-supervised clustering. In *DSN*, pages 307–316. IEEE, 2009.

[179] Li Xiao, Xiaodong Zhang, and Zhichen Xu. On reliable and scalable peer-to-peer web document sharing. In *Proceedings of the 16th International Parallel and Distributed Processing Symposium (IPDPS '02)*, page 228, Washington, DC, USA, 2002. IEEE Computer Society.

[180] Li Xiong and Ling Liu. Peertrust: Supporting reputation-based trust for peer-to-peer electronic communities. *IEEE Trans. on Knowl. and Data Eng.*, 16:843–857, July 2004.

[181] Ruishan Zhang, Xinyuan Wang, Xiaohui Yang, and Xuxian Jiang. Billing attacks on sip-based voip systems. In *Proceedings of the first USENIX workshop on Offensive Technologies*, pages 4:1–4:8, Berkeley, CA, USA, 2007. USENIX Association.

[182] Ben Y. Zhao, John Kubiatowicz, Anthony D. Joseph, Ben Y. Zhao, John Kubiatowicz, and Anthony D. Joseph. Tapestry: an infrastructure for fault-tolerant wide-area location and routing. Technical report, UC Berkeley, 2001.

[183] Yingwu Zhu and Haiying Shen. Trustcode: P2p reputation-based trust management using network coding. In *Proceedings of the 15th international conference on High performance computing*, HiPC'08, pages 378–389, Berlin, Heidelberg, 2008. Springer-Verlag.

Appendix A P2P HTTP cache advertisement algorithm

Following is a meta-code listing for the P2P HTTP caching algorithm discussed in Chapter 6.2.1.

```
class Advertisement:
    """An advertisement for objects originating from a single host."""

    def __init__(self, cache):
        self.requests = []
        self.cache = cache

        # The oldest unpublished object
        self.oldest = 0
        # The furthest point in time we have valid objects
        self.valid_until = 0

    def publish(self, time):
        """Publishes an advertisement"""

        # Record the accessibility delay
        for request, request_time in self.requests:

            # Include only objects that are still valid
            if request_time + request.expire > time:
                self.cache.published.items += 1
                self.cache.published.delay += time - request_time

        self.last_publish = time
        self.oldest = 0

    def add_event(self, request, time):
        """Adds an object to the advertisement"""

        # Record the pause since the last request
        pause = time - self.last_update
        self.last_update = time

        # Check whether this was a 'miss'
        if pause > self.cache.PUBLISH_DELAY and
           pause < self.cache.AVERAGE_DELAY:
            self.cache.ad_miss = True

        # Update the time of the longest valid object
        if request.expire + time > self.valid_until:
            self.valid_until = request.expire + time

        self.requests.add(request, time)

class Cache:
    """The cache - keeps track of all advertisements"""

    def __init__(self, avg = 300, max = 15*60):
        self.ad_miss = False
        self.published_count = 0
        self.published.delay = 0
        self.published.items = 0
        self.advertisements = {}

        # The average delay we strive to
        self.AVERAGE_DELAY = avg
        # The maximum delay limit
        self.MAX_DELAY = max

        # Initially, the publish delay is set to the strived-to average
        self.PUBLISH_DELAY = self.AVERAGE_DELAY

        # By how much we should increase / decrease the timeouts
        self.PUBLISH_INCREASE = 0.1

    def check(self, time):
        """Checks whether we should publish or adjust the timouts"""

        # For each host, check whether we should publish an advertisement
        for ad in self.advertisements:

            # if there has been activity, we have valid objects
            # and a timeout has passed, publish
            if ad.last_update > ad.last_publish and ad.valid_until > time:
                if (ad.last_update + self.PUBLISH_DELAY) < time
                or (ad.oldest + self.MAX_DELAY) < time:
                    ad.publish(time)
                    self.published_count += 1

        # Adjust the timeouts based on the average accessibility delay
        if self.published_count > 10:
            acc_delay = self.published.delay / self.published.items
            adjustment = 0

            # Decrease timeout if we are over the average
            if acc_delay > self.AVERAGE_DELAY:

                # The decrease is proportional
                diff = acc_delay - self.AVERAGE_DELAY
                adjustment = (diff * self.PUBLISH_DELAY)) / acc_delay
            else:

                # Otherwise, decrease the timeout, unless..
                adjustment = self.PUBLISH_INCREASE * self.PUBLISH_DELAY

                # ..we had a miss: increase the timeout
                if self.ad_miss:
                    adjustment = -adjustment

            self.PUBLISH_DELAY -= adjustment

            # Re-set the counters
            self.ad_miss = False
            self.published_count = 0
            self.published.delay = 0
            self.published.items = 0

    def process_event(self, request, time):

        # Add each event to the host-specific advertisements
        ad = self.advertisements[request.host]
        if ad is None:
            ad = Ad(self)
            self.advertisements[request.host] = ad

        ad.add_event(request, time)


# The cache
cache = Cache()

# Point of entry for the cache simulator. The all_events array
# contains all requests, ordered by time.
all_events = load("browsing.log")
for time in range(1, all_events.max_time):
    for request in all_events.requests_at[time]:
        cache.process_event(request , time)

    # Each second, check whether we should publish or adjust the timeout
    check.cache(time)
```

Aalto-DD 14/2015

ISBN 978-952-60-6063-7 (printed)
ISBN 978-952-60-6064-4 (pdf)
ISSN-L 1799-4934
ISSN 1799-4934 (printed)
ISSN 1799-4942 (pdf)

Aalto University
School of Science
Department of Computer Science
www.aalto.fi

BUSINESS +
ECONOMY

ART +
DESIGN +
ARCHITECTURE

SCIENCE +
TECHNOLOGY

CROSSOVER

**DOCTORAL
DISSERTATIONS**