IMPROVING THE CAPABILITIES OF DISTRIBUTED COLLABORATIVE INTRUSION DETECTION SYSTEMS USING MACHINE LEARNING

CARLOS GARCÍA CORDERO

Dissertation
Zur Erlangung des akademischen Grades
Doktor-Ingenieur (Dr.-Ing.)

genehmigte Dissertationsschrift in englischer Sprache
von MSc. Carlos García Cordero
aus Darmstadt, Germany
geboren in Mexiko Stadt, Mexiko

Erstreferent: Prof. Dr. Max Mühlhäuser
Korreferent: Prof. Dr. René Mayrhofer
Korreferent: Prof. Dr. Sascha Hauke

Tag der Einreichung: 6 May 2019
Tag der Prüfung: 14 June 2019

Fachgebiet Telekooperation
Fachbereich Informatik
Technische Universität Darmstadt
Hochschulkennziffer D-17
Darmstadt, 2019

August 21, 2019 — version 1.2

Carlos García Cordero:
Improving the Capabilities of Distributed Collaborative Intrusion Detection Systems using Machine Learning

Darmstadt, Technische Universität Darmstadt
Jahr der Veröffentlichung der Dissertation auf TUprints: 2019
URN: urn:nbn:de:tuda-tuprints-90033
Tag der Prüfung: 14.06.2019

I can't help but ask, one day many years later, when you find your previous awareness, cognition and choices are all wrong, will you keep going along the wrong path or reject yourself?

— *Gu Li* after playing against *AlphaGo*

SYNOPSIS

The impact of computer networks on modern society cannot be estimated. Arguably, computer networks are one of the core enablers of the contemporary world. Large computer networks are essential tools which drive our economy, critical infrastructure, education and entertainment. Due to their ubiquitousness and importance, it is reasonable to assume that security is an intrinsic aspect of their design. Yet, due to how networks developed, the security of this communication medium is still an outstanding issue.

Proactive and reactive security mechanisms exist to cope with the security problems that arise when computer networks are used. Proactive mechanisms attempt to prevent malicious activity in a network. Prevention alone, however, is not sufficient: it is imprudent to assume that security cannot be bypassed. Reactive mechanisms are responsible for finding malicious activity that circumvents proactive security mechanisms. The most emblematic reactive mechanism for detecting intrusions in a network is known as a Network Intrusion Detection System (NIDS).

Large networks represent immense attack surfaces where malicious actors can conceal their intentions by distributing their activities. A single NIDS needs to process massive quantities of traffic to discover malicious distributed activities. As individual NIDSs have limited resources and a narrow monitoring scope, large networks need to employ multiple NIDSs. Coordinating the detection efforts of NIDSs is not a trivial task and, as a result, Collaborative Intrusion Detection Systems (CIDSs) were conceived. A CIDS is a group of NIDSs that collaborate to exchange information that enables them to detect distributed malicious activities. CIDSs may coordinate NIDSs using different communication overlays.

From among the different communication overlays a CIDSs may use, a distributed one promises the most. Distributed overlays are scalable, dynamic, resilient and do not have a single point of failure. Distributed CIDSs, i. e., those using distributed overlays, are preferred in theory, yet not often deployed in practice. Several open issues exist that constraint the use of CIDSs in practice.

In this thesis, we propose solutions to address some of the outstanding issues that prevent distributed CIDSs from becoming viable in practice. Our contributions rely on diverse Machine Learning (ML) techniques and concepts to solve these issues. The thesis is structured around five main contributions, each developed within a dedicated chapter. Our specific contributions are as follows.

DATASET GENERATION We survey the intrusion detection research field to analyze and categorize the datasets that are used to develop, compare, and test NIDSs as well as CIDSs. From the defects we found in the datasets, we develop a classification of dataset defects. With our classification of dataset issues, we develop concepts to create suitable datasets for training and testing ML based NIDSs and CIDSs. With our concepts, we injects synthetic attacks into real background traffic. The generated attacks replicate the properties of the background traffic to make attacks as indistinguishable as they can be from real traffic.

INTRUSION DETECTION We develop an anomaly-based NIDS capable of overcoming some of the limitations that NIDSs have when they are used in large networks. Our anomaly-based NIDS leverages autoencoders and dropout to create models of normality that accurately describe the behavior of large networks. Our NIDS scales to the number of analyzed features, can learn adequate normality models even when anomalies are present in the learning data, operates in real time, and is accurate with only minimal false positives.

COMMUNITY FORMATION We formulate concepts to build communities of NIDSs, coined community-based CIDSs, that implement centralized ML algorithms in a distributed environment. Community-based CIDSs detect distributed attacks through the use of ensemble learning. Ensemble learning is used to combine local ML models created by different communities to detect network-wide attacks that individual communities would otherwise struggle to detect.

INFORMATION DISSEMINATION We design a dissemination strategy specific to CIDSs. The strategy enables NIDSs to efficiently disseminate information to discover and infer when similar network events take place, potentially uncovering distributed attacks. In contrast to other dissemination strategies, our strategy efficiently encodes, aggregates, correlates, and shares network features while minimizing network overhead. We use Sketches to aggregate data and Bayesian Networks to deduce new information from the aggregation process.

COLLUSION DETECTION We devise an evidence-based trust mechanism that detects if the NIDSs of a CIDS are acting honestly, according to the goals of the CIDS, or dishonestly. The trust mechanism uses the reliability of the sensors and Bayesian-like estimators to compute trust scores. From the trust scores, our mechanism is designed to detect not only single dishonest NIDSs but multiple coalitions of dishonest ones. A coalition is a coordinated group of dishonest NIDSs that lie to boost their trust scores, and to reduce the trust scores of others outside the group.

ZUSAMMENFASSUNG

Die Auswirkungen von Computernetzwerken auf die moderne Gesellschaft lassen sich nicht abschätzen. Zweifellos sind Computernetzwerke einer der wichtigsten Faktoren in der heutigen Welt. Große Computernetzwerke sind unverzichtbare Werkzeuge, die unsere Wirtschaft, kritische Infrastruktur, Bildung und Unterhaltung antreiben. Aufgrund ihrer Allgegenwärtigkeit und Bedeutung ist es sinnvoll anzunehmen, dass Sicherheit ein wesentlicher Aspekt ihres Designs ist. Doch aufgrund der Entwicklung der Netzwerke ist die Sicherheit dieses Kommunikationsmediums noch ein offenes Thema.

Zur Bewältigung der Sicherheitsprobleme, die bei der Nutzung von Computernetzwerken auftreten, werden aktuell proaktive und reaktive Sicherheitsmechanismen eingesetzt. Proaktive Mechanismen versuchen, böswillige Aktivitäten in einem Netzwerk zu verhindern. Prävention allein reicht jedoch nicht aus: Es ist leichtsinnig anzunehmen, dass Sicherheit nicht umgangen werden kann. Reaktive Mechanismen sind dafür verantwortlich, gerade die Aktivitäten zu entdecken, die proaktive Sicherheitsmechanismen umgehen. Der wohl bekannteste reaktive Mechanismus zur Erkennung von Eindringlingen in einem Netzwerk ist bekannt als NIDS.

Große Netzwerke stellen immense Angriffsflächen dar, deren Größe es böswilligen Akteuren ermöglicht, ihre Absichten durch die Verteilung ihrer Aktivitäten zu verbergen. Ein einzelnes NIDS muss große Mengen an Datenverkehr verarbeiten, um bösartige verteilte Aktivitäten zu entdecken. Da einzelne NIDSs nur über begrenzte Ressourcen und einen eingeschränkten Überwachungsradius verfügen, müssen große Netzwerke mehrere NIDSs einsetzen. Da die Koordination der Erkennungsbemühungen von NIDSs keine triviale Aufgabe darstellt, wurden als Lösung CIDSs konzipiert. Ein CIDS besteht aus einer Gruppe von NIDSs, die zusammenarbeiten, um Informationen auszutauschen, die es ihnen ermöglichen, verteilte bösartige Aktivitäten zu erkennen. NIDSs können durch CIDSs unter Verwendung verschiedener Kommunikationsüberlagerungen koordiniert werden.

Aus den verschiedenen Kommunikations-Overlays, die ein CIDSs verwenden kann, ist ein verteilter Ansatz der vielversprechenste. Verteilte Overlays sind skalierbar, dynamisch, resilient und haben keinen zentralen Schwachpunkt. Verteilte CIDSs, die verteilte Overlays verwenden, werden in der Theorie bevorzugt, aber in der Praxis nicht häufig eingesetzt. Es gibt mehrere offene Fragen, die den Einsatz von CIDSs in der Praxis einschränken.

In dieser Arbeit schlagen wir Lösungen vor, mit dem Ziel, einige der noch offenen Fragen zu addressieren, die verhindern, dass verteilte CIDSs in der Praxis nutzbar werden. Unsere Beiträge basieren

auf verschiedenen ML Techniken und Konzepten, um dieses Ziel zu erreichen. Die Arbeit beinhaltet fünf Hauptbeiträge, die jeweils in einem eigenen Kapitel beschreiben werden. Unsere spezifischen Beiträge lauten wie folgt.

DATENSATZERSTELLUNG Wir untersuchen das Feld wissenschaftlicher Arbeiten zu Intrusion Detection Systems (IDSs), um die Datensätze zu analysieren und zu kategorisieren, die zur Entwicklung, zum Vergleich und zum Testen von NIDSs und CIDSs verwendet werden. Aus den Defiziten, die wir in den Datensätzen gefunden haben, entwickeln wir eine Klassifizierung für Datensatzprobleme. Mit unserer Klassifizierung von Datensatzproblemen entwickeln wir Konzepte zur Erstellung geeigneter Datensätze zum Trainieren und Testen ML-basierter NIDSs und CIDSs. Mit unseren Konzepten injizieren wir synthetische Angriffe in realen Hintergrunddatenverkehr. Die erzeugten Angriffe replizieren die Eigenschaften des Hintergrunddatenverkehrs, um Angriffe dadurch von echtem Datenverkehr ununterscheidbar zu machen.

EINBRUCHSERKENNUNG Wir schlagen ein anomaliebasiertes NIDS vor, das in der Lage ist, einige der Einschränkungen von NIDSs zu überwinden, die auftreten, wenn diese in großen Netzwerken eingesetzt werden. Unser anomaliebasiertes NIDS nutzt Autoencoder und Dropout, um Modelle der Normalität zu erstellen, die das Verhalten großer Netzwerke akkurat beschreiben. Unser NIDS skaliert hinsichtlich der Anzahl analysierter Merkmale, ist resilient gegenüber dem Lernen auf Datensätzen, die Angriffe beinhalten, arbeitet in Echtzeit und hat eine genaue Erkennungsrate bei minimaler Anzahl von Fehlalarmen.

ERSTELLUNG VON GEMEINSCHAFTEN Wir formulieren Konzepte zum Aufbau von Gemeinschaften von NIDSs, genannt gemeinschaftsbasierte CIDSs. Diese implementieren zentralisierte ML Algorithmen in einer verteilten Umgebung. Gemeinschaftsbasierte CIDSs erkennen verteilte Angriffe durch den Einsatz von Ensemble Learning. Ensemble Learning wird verwendet, um lokale ML Modelle zu kombinieren, die von verschiedenen Gemeinschaften erstellt wurden, um netzwerkweite Angriffe zu erkennen, die einzelne Gemeinschaften sonst nur schwer erkennen würden.

INFORMATIONSVERBREITUNG Wir entwickeln eine Verbreitungsstrategie, die speziell auf CIDSs zugeschnitten ist. Die Strategie ermöglicht es NIDSs, Informationen effizient zu verbreiten, um ähnliche Netzwerkereignisse zu erkennen und daraus Rückschlüsse zu ziehen, um potenziell verteilte Angriffe aufzudecken. Im Gegensatz zu anderen Verbreitungstechniken kodiert, aggregiert, korreliert und

teilt unsere Verbreitungsstrategie Netzwerkmerkmale effizient und minimiert gleichzeitig den Netzwerk-Overhead. Bayes'sche Netzwerke und Sketches dienen hierbei als Hebelmechanismen.

KOLLUSIONSERKENNUNG Wir entwickeln einen evidenzbasierten Vertrauensmechanismus, der erkennt, ob die NIDSs einer CIDS ehrlich, nach den Zielen der CIDS, oder unehrlich handeln. Der Vertrauensmechanismus nutzt die Zuverlässigkeit der Sensoren und orientiert sich an Bayes'schen Schätzern, um Vertrauenswerte zu berechnen. Der Mechanismus wurde entwickelt, um nicht nur einzelne unehrliche NIDSs, sondern auch mehrere Koalitionen von unehrlicher NIDSs zu erkennen. Eine Koalition ist eine koordinierte Gruppe von unehrlichen NIDSs. Die NIDSs einer Koalition lügen, um ihre Vertrauenszahlen zu erhöhen und die Vertrauenszahlen anderer außerhalb der Koalition zu reduzieren.

Mephistopheles: "[Some things] lie outside the boundaries that words can address; and man can only grasp those thoughts which language can express."
Faust: "What? Do you mean that words are greater yet than man?"
Mephistopheles: "Indeed they are."
Faust: "Then what of longing? Or affection, pain and grief? I can't describe these, yet I know they are in my breast. What are they?"
Mephistopheles: "Without substance, as mist is."
Faust: "In that case man is only air as well!"

— From the movie "Faust", by Jan Švankmajer

The 1994 Faust movie by Jan Švankmajer plays with the rhetoric that words (language) are the enablers of thought. The argumentation that language is essential to thought has been a long-standing debate with many in favor (e.g., Hegel, Nietzsche) and against (e.g., Rousseau, Bergson). Both parties, however, (arguably) agree that language is the best tool to preserve thought. In the context of modern science, the written word is that which enables us, scientists, to develop, present and share our scientific thoughts. Acknowledging the importance of the written word enabled me to sit down long nights to develop the contents of the present work (in spite of all the hardships).

In the novel of Goethe, Dr. Faust struggles to find and develop novel scientific thoughts and, due to his frustration, sells his soul to Mephistopheles. In the song "Faustian Echoes", Agalloch expresses what this relatable frustration must have been to Faust:

O growing Moon, didst thou but shine
A last time on this pain of mine
Behind this desk how oft have I
At midnight seen thee rising high
O'er book and paper I bend
Thou didst appear, o mournful friend

— From the song "Faustian Echoes", by Agalloch

Instead of relying on Mephistopheles' help, like Faust did, I relied on the support of many around me to cope with the inherent frustrations associated with the development of novel scientific work.

Prof. Mühlhäuser, you have been a great supervisor and an extraordinary boss. The freedom you have given me to carry out my research made me a better researcher. Every discussion I had with you, no matter the subject, always brought up interesting points that challenged

my perception of things. *"Where do you believe we think?"*, you asked me once. *"In the brain of course"*, I replied. *"I think it was my stomach, and not my brain, that made the decision to go in and disturb that lecture while I was pulling the Ph.D. celebration wagon"*. Overall, my time working in the Telecooperation Lab was stressful but rewarding and fun. Much of the fun was due to the great environment brought about by my colleagues. Thank you Aidmar, Andrea, Fabio, Florian, Julien, Jörg, Leon, Manolis, Mathias, Nikos, Rolf, Sascha, Shankar, Sheikh, Tim, and all the others for the support and your patience.

To all my family, specially: my mom, dad, sister, aunts, uncles and cousin; thank you for letting me know that you are around me despite the distance. To all my friends, in Darmstadt, Mexico and all over the world, thank you for the great times we have had and the times that we have yet to live! Yetty, do not stop sending messages with that ever-present chant of yours! Myriam, every discussion I have with you is amazing. What an amazing friend you are. I am always looking up to you as inspiration. Oso, when are we cooking together again? Flaca, we should listen and compose post-rock music together! Ilaine, let us chat more often! Muerto, when are we jamming together? Paulina, let us go grab dinner some time soon. Mariana, we should cook mole and Mexican food again! Angie, I promise to visit you more often in Edinburgh (or wherever you are). Pepe, have you heard the latest album of Nargaroth!? It is amazing! Fernando, let us climb once a week again and play as many board games as possible! Max, middle earth was too peaceful for far too long. The hunt for the ring shall continue! Rahul, now that we both have experienced it, let us compose "depressive thesis writing black metal"! Oh, and I just discovered a new Russian Circles album is out! Gustavo, when is the next band practice taking place? Are we finally learning how to play "Babe I'm gonna leave you"?

Vanessa, let this that is written here be a remembrance of the history that was and is. If they ask, we will say that, indeed, it was true; for what is true but that which we perceive?

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACRONYMS

ANIDS	Anomaly-based Network Intrusion Detection Systems
API	Application Programming Interface
AS	Anomaly Score
CAIDA	Center for Applied Internet Data Analysis
CDN	Content Distribution Network
CDX	Cyber Defense Exercise
CIDS	Collaborative Intrusion Detection System
CPT	Conditional Probability Table
DAG	Directed Acyclic Graph
DARPA	Defense Advanced Research Projects Agency
DDoS	Distributed Denial of Service
DHT	Distributed Hash Table
DLL	Dynamic Link Library
DoS	Denial of Service
FLAME	Flow-Level Anomaly Modeling Engine
FOSS	Free and Open Source Software
GAN	Generative Adversarial Network
GMM	Gaussian Mixture Model
IANA	Internet Assigned Numbers Authority
ICSI	International Computer Science Institute
ID2T	Intrusion Detection Dataset Toolkit
IDS	Intrusion Detection System
IMPACT	Information Marketplace for Policy and Analysis of Cyber-risk & Trust
IRSC	Indian River State College
IoT	Internet of Things
JSD	Jensen-Shannon Divergence

LBNL	Lawrence Berkeley National Laboratory
LERAD	Learning Rules for Anomaly Detection
MLE	Maximum Likelihood Estimation
ML	Machine Learning
MSS	Maximum Segment Size
NAT	Network Address Translation
NIC	Network Interface Card
NIDS	Network Intrusion Detection System
P2P	Peer to Peer
PCAP	Packet Capture
PCA	Principal Components Analysis
PC	Principal Component
PDF	Probability Density Function
PDS	Probabilistic Data Structure
PDS	Probabilistic Data Structure
PHAD	Packet Header Anomaly Detector
PMF	Probability Mass Function
RNN	Replicator Neural Network
SDN	Software Defined Networking
SGD	Stochastic Gradient Descent
SNIDS	Signature-based Network Intrusion Detection Systems
SPoF	Single Point of Failure
TIDED	Testing Intrusion Detection Datasets
TTL	Time to Live
ToS	Type of Service
VPN	Virtual Private Network
i.i.d.	independent and identically distributed

1 INTRODUCTION

The entanglement between computer networks and modern society has increased in such a way that without networks the "modern" qualifier in "modern society" could be dismissed. We interact with networks, whether directly or indirectly, to carry out both simple and complicated activities. On the Internet, we conduct business, find entertainment, share experiences and interact with others on a regular basis. In 2017, for example, it is estimated that close to 80 percent of all adults in Europe and North-America used the Internet almost every day [*The Connected Consumer Survey* 2017; Wagner, 2018]. With such a prolific use, security should be one of the core aspects of networks. At present, nonetheless, network security is an afterthought rather than the outcome of careful design.

network security is an afterthought

When networks of computers were first put together in the seventies, the security of the communication channels was not a core concern [Oppliger, 2001]. One of the first large-scale computer network, Arpanet, was exclusive to a relatively small population of trusted users, i. e., the military and few researchers. Arpanet's design and goals were custom-made to move information quickly and reliably. Network security only consisted in defending against external threats rather than its own users [Timberg, 2015]. As security issues started to proliferate within Arpanet, security began to be patched on top of already existing components. Arpanet eventually became what is now the Internet and the core communication stack of today (e. g., TCP/IP). Modern networks inherit old design principles that, to this day, make them susceptible to attacks of their own users. Many old design decisions are questionable, causing more troubles than benefits. For example, Bärwolff [2010] studied the decision of making intermediary network infrastructure responsible for flow control, error control and resource management instead of end-hosts.

networks are designed to move information

network security is a patch

Due to the ubiquitousness and importance of large networks, protecting them is indispensable, yet the task is challenging: Networks keep growing exponentially just as they continue to increase their transportation capacity [Inacio et al., 2010]. The combination of their size and our dependence to them creates immense attack surfaces that malicious users seem to exploit uncontestedly. Detecting attacks within large networks is challenging and is further exacerbated by the fact that coordinated attacks are becoming the norm. Network operators and data centers, for example, consider coordinated Distributed Denial of Service (DDoS) attacks as the biggest threat they face [*Worldwide Infrastructure Security Report* 2014]. Some 30 percent

immense attack surfaces

of UK companies estimate that they would lose £10,000 or more for each hour of a DDoS attack [Neustar, 2014]. Despite the disruptive capability of coordinated attacks such as DDoSs, countermeasures do not appear to be in place. We conclude this from the fact that, in the span of 6 months from October 2017 to March 2018, the number of overall DDoS attacks on the Internet more than doubled [Verisign, 2018].

separation of concerns

Network security has largely been incorrectly regarded as a problem that can be solved at the edge (e. g., [Markham et al., 2001]). At the edge, where end users operate, we can easily establish security countermeasures through *separation of concerns*: Each end point is responsible for their own security. We argue, however, that this approach is incapable of detecting network-wide threats. Separation of concerns is a veil that gives a false sense of security and enables coordinated malicious individuals to disguise their actions through distribution. In order to detect collaborative attacks, we require collaborative defenses. This is especially true if we wish to detect intrusions in large networks.

security at the edge is not enough

1.1 INTRUSION DETECTION IN LARGE NETWORKS

The constant growth of sophisticated, distributed and coordinated attacks poses a serious threat to users and infrastructures alike. Besides focusing on financial gain at the expense of users, malicious individuals are slowly shifting focus towards the disruption of critical infrastructure (e. g., state sponsored attacks) [Mee et al., 2018]. To actively counteract these threats, network operators employ *proactive security* systems such as firewalls, anti-virus scanners, Virtual Private Networks (VPNs), public key authentication or content access policies. These proactive security measures, now common in every network, restrain malicious activity without being fully capable of preventing it. *Reactive security* mechanisms are a second line of defense responsible for finding malicious activity that may slip through proactive security mechanisms. Intrusion Detection Systems (IDSs) are key reactive mechanism extensively studied and used in many domains [Butun et al., 2004; Lazarevic, Kumar, et al., 2005; Mitchell et al., 2014].

proactive security

reactive security

intrusion detection system

IDSs monitor a host or a network for signs of undesired activities, often pointing to security violations. An IDS that focuses on monitoring network activity is known as a Network Intrusion Detection System (NIDS). Network intrusion detection can be carried out through *misuse analysis* or *anomaly detection*. Misuse analysis is the process of recognizing previously seen malicious traffic using signatures. Anomaly detection, instead, first models a network's normal behavior and then discovers behavior that does not conform to the model. Misuse or anomaly detection can be applied at the packet level, network flow level or both. At the packet level, individual network packets are the subject of analysis. Within small and medium sized networks, intru-

network intrusion detection system

sion detection at the packet level is taxing, yet achievable. In large networks, however, such a luxury is not available due to the number of generated packets. Instead, large networks rely on intrusion detection at the network flow level.

The state of the art for detecting intrusions embedded in large traffic quantities relies on NIDSs examining network flows in search for intrusions. *Network flows* are collections of features that relate to the packets exchanged between two network devices. Network flows consist of communication and packet aggregation statistics. Some statistics include the number of exchanged bytes, the duration of communication and the packets sent, among others. A formal and more detail explanation of network flows is covered in Section 4.2.1.

network flows

The current trend followed by organizations to detect intruders in large networks is to collect network flows in a central dataset which an NIDS can analyze [Sperotto and Pras, 2010]. Distributed flow exporters are responsible for monitoring a network and constructing flows. All flows are sent to flow collectors which in turn store the flows in a central location to create a dataset. An NIDS examines this dataset searching for intrusions. This approach is effective but has several architectural disadvantages. A centralized architecture contains a Single Point of Failure (SPoF) and has limited scalability as a single NIDS must posses ample computational resources. Furthermore, with such an architecture, a single NIDS is given full access to all information which may pose a privacy risk. These conditions are prohibitive, especially when involving multiple organizations with different domain boundaries. To overcome these issues, we need a distributed and collaborative environment. In collaborative environments, autonomous participants can share information with different degrees of granularity and share computational resources. Collaborative Intrusion Detection Systems (CIDSs) address these issues and provide the theoretical foundations by which we can detect single and coordinated intruders in large networks.

1.2 AN OVERVIEW ON CIDS

CIDSs are collections of autonomous NIDS[1] sensors that together exchange information to enable the detection of collaborative and distributed network attacks. Sensors are responsible for performing intrusion detection on top of network traffic they collect. A communication overlay connects sensors together, possibly taking into account communication restrictions (e. g., [Vasilomanolakis, Krugl, et al., 2016]), to enable information sharing. One or more analyzers are

1 CIDSs may also be formed using host-based IDSs (see Section 2.2). Throughout this thesis, however, we only consider CIDSs composed of NIDSs. This consideration does not hamper the applicability of our contributions if host-based IDSs would instead be used.

then responsible for identifying, through correlation and aggregation of shared data, common events experienced by different sensors.

Until recently, centralized, hierarchical or distributed CIDSs were not considered to have a standardized or established set of components. In the survey work of Vasilomanolakis, Karuppayah, et al. [2015], they identify five components that together form a general CIDS architecture. As a reference only[2], we show in Figure 1.1 the architectural components that make up a CIDS. Each component operates like a black box[3] with respect to all other components, only using as inputs the output of those components directly below or next to it. This architectural quality allows us to address individual issues within each component to develop more robust and capable CIDSs. For these reasons, we reference this architecture rather than imitating other CIDS architectures (e. g., [Yu et al., 2005; Chenfeng Vincent Zhou et al., 2010a]).

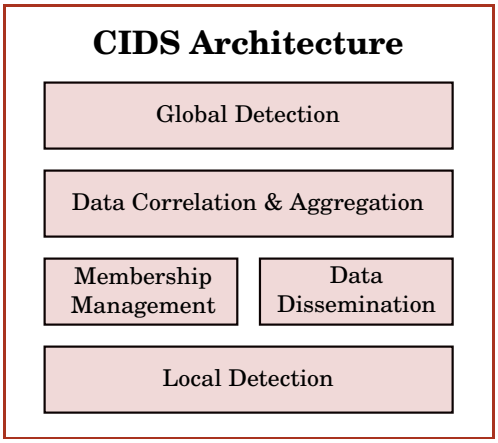


Figure 1.1: The five architectural components that make up a CIDS.

We use the aforementioned modular CIDS architecture to orient the reader with respect to our work: Each of our contributions explicitly addresses issues that lie within one or more of the five components of the CIDS architecture model we reference. Each of our contribution chapters (i. e., from Chapter 3 to 7) includes an overview that uses Figure 1.1 to highlights the CIDS components the chapter addresses.

1.3 OPEN ISSUES WITHIN DISTRIBUTED CIDSS

CIDSs carry out their work using multiple NIDSs. Depending on how NIDSs organize, a CIDS is classified as centralized, hierarchical or distributed. Distributed CIDSs are the most promising of the three from

2 We give a comprehensive and detailed explanation of each component later in Section 2.3.3.

3 The term "black box" is used in the Machine Learning (ML) field to refer to a process that is not transparent to the viewer and may be understood only in terms of its inputs and outputs.

a theoretical perspective but suffer from several issues which hamper their distributed capabilities in practice. We identify five pressing issues that without adequate solutions make fully distributed CIDSs only plausible in theory.

THE DATASET ISSUE The intrusion detection field has the long standing issue of lacking datasets [Catania et al., 2012]. Yet, standard and open datasets are the key to develop, evaluate and compare CIDSs. Datasets play an especially important role in the development of CIDS that are based on ML. Nevertheless, there is no single dataset or tool that researchers in the CIDS field can easily use. Those available datasets are either outdated [Tavallaee et al., 2009], lack ground truth (e. g., [Fontugne et al., 2010]), contain known deficiencies (e. g., [Lippmann et al., 1999]) or only reflect highly specialized and limited scenarios (e. g., [CAIDA, 2018]). Without adequate datasets, the community lacks one of the basic tools needed to accelerate the pace by which the CIDS field advances. This is especially true when ML is involved.

THE SCALABLE ANOMALY DETECTION ISSUE In the past, misuse or signature-based NIDSs were heavily used due to their accuracy and effectiveness [Axelsson, 1998]. Today, network misuse analysis is no longer as effective as before due to large attack surfaces, the polymorphic nature of attacks, the widespread usage of encryption and the amount of new attacks surfacing every day. Therefore, anomaly-based NIDSs need to be prioritized. Anomaly detection relies on creating normality models which can then identify abnormal behavior. Building scalable normality models with the amount of data observed in large networks is challenging. Furthermore, anomaly detection is known to have, in comparison to misuse detection, high rates of false alarms. When analyzing large traffic quantities, a small false alarm rate still translates to a large number of alerts that an analyst has to study.

THE COLLABORATION ISSUE CIDSs usually operate at what we term the *alarm level*. At this level, NIDSs first perform intrusion detection in isolation and then share the alarms they yield with others. Distributed attacks are discovered from the aggregation and collection of alarms. We recognize an alternative level of CIDS operation which we coin the *detection level*. Instead of performing intrusion detection in isolation, at the detection level, NIDSs collaborate to build distributed Machine Learning (ML) models that can be used to perform intrusion detection. By building models in collaboration with others, subtle distributed and collaborative attacks may be detected earlier and more accurately.

alarm level

detection level

ML models are typically learned using centralized mechanisms. In distributed collaborative environments, however, centralized compo-

nents degrade the quality of the system. Many effective centralized mechanisms already exist. Instead of rebuilding these mechanisms from scratch to operate within a distributed environment, collaboration should be leveraged to find ways to join several central models together to create distributed ones. Without solving this issue, distributed CIDSs require especially tailored algorithms, becoming an issue on its own.

THE DISSEMINATION ISSUE The NIDS sensors of a CIDSs establish collaboration by means of information exchange. When a communication overlay is not defined in advance, exchanging information becomes difficult, especially if we set the goal of minimizing communication overhead. Therefore, in distributed environments, information should be exchanged using a carefully designed dissemination mechanism. As designing and incorporating such a mechanism is complicated and time consuming, CIDS designers take the dissemination mechanism for granted (i. e., communication between sensors just happens), or use inefficient but easy to implement solutions (e. g., network flooding) [Vasilomanolakis, Karuppayah, et al., 2015].

dissemination techniques

No dissemination mechanism exists that is specifically customized to CIDSs. The typical dissemination techniques used within CIDSs are flooding, gossiping, publish-subscribe and centralized communication [Vasilomanolakis, Karuppayah, et al., 2015]. Flooding techniques incur in high communication overhead costs and do not scale well. Gossiping techniques, also known as epidemic techniques [Gupta et al., 2010], lower the communication overhead at the expense of unreliable data delivery. Publish-subscribe suffers from the high costs of managing and maintaining an overlay. Centralized communication introduces a SPoF, among many other problems. CIDSs require a dissemination mechanism that scales well, reduces communication overhead and is tailored to the dissemination of the type of information NIDSs need.

THE COLLUSION ISSUE CIDSs are meant to protect large network infrastructures against attacks. Yet, most CIDSs do not protect themselves against insider attacks [Chenfeng Vincent Zhou et al., 2010b]. The vast majority of work in the CIDS field assumes that collaborating NIDSs are honest and trustworthy. This assumption does not adequately hold in real-world settings. Things are changing, however, as researchers start to propose trust-based mechanisms to detect dishonest CIDS members (e. g., [C. J. Fung, J. Zhang, et al., 2011]). This is a positive step forward but more work is required in the direction of detecting collusion. In the CIDS context, *collusion* is an agreement between dishonest NIDSs to secretly deceive the system in their favor.

collusion in CIDSs

1.4 RESEARCH GOALS AND OBJECTIVES

The goal of this thesis is to propose algorithms, systems and concepts to improve the capabilities of fully distributed CIDSs with the help of ML. To achieve our goal, we focus on accomplishing different objectives, all of which relate to the aforementioned open issues within distributed CIDSs. For each open issue, we devote a chapter of this thesis to its study and the proposal of potential solutions. In the following, we state the high level objectives on which we focus, to which open issue the objectives relate, and the chapter where the objectives are addressed.

- To address the *the dataset issue*, in Chapter 3, we set two objectives. The first objective is to survey available datasets to determine their usefulness for evaluating and developing NIDSs. The second objective is to develop a tool capable of creating suitable datasets for evaluating NIDSs.
- To approach *the scalable anomaly detection issue*, in Chapter 4, we establish the objective of developing an unsupervised anomaly detection technique that scales to large networks.
- In Chapter 5, to tackle *the collaboration issue*, we develop the theory behind the formation of NIDS communities within a CIDS. Building communities serves us as a mean to achieve two different objectives. The first objective is to propose methods for using centralized ML algorithms within a distributed environment (i. e., a community). We wish to develop a mechanism capable of balancing the communication overhead incurred within a community with the detection accuracy achieved by the community. The second objective is to develop the concept of collaboration at the *detection level* which enables us to build distributed ML models within CIDSs.
- To undertake *the dissemination issue*, we establish two objectives in Chapter 6. The first objective is to develop a dissemination strategy tailored to the needs of CIDSs. The second objective is to enable collaborating NIDS sensors that use such a dissemination strategy to work under uncertainty. Working under uncertainty implies that sensors can infer from partial observations and do not need to wait for the dissemination mechanism to converge before using shared information.
- Finally, in Chapter 7, we work on *the collusion issue* by establishing three objectives. The first objective is to propose a fast and efficient system capable of detecting dishonest CIDSs sensors. The second objective is to enable CIDSs to detect one ore more groups of colluding sensors as long as the colluders do not overwhelm the honest sensors. The third objective is to detect dishonest sensors and colluders that choose to act honestly from time to time to fool the defensive mechanisms of a CIDS.

1.5 SCIENTIFIC CONTRIBUTIONS

We leverage several ML techniques such as neural networks, probabilistic inference, clustering, Gaussian Mixture Models (GMMs) and ensemble learning throughout this thesis to improve the capabilities of distributed CIDSs. We further develop concepts and evaluate ideas to advance the field of distributed CIDSs and aid researchers in the development of tools. Overall, the thesis consists of five core chapters, each having the following contributions:

- Dataset Generation (Chapter 3)
  - This chapter provides a comprehensive and extensive survey on the datasets available for the evaluation and development of NIDSs. In our survey, we organize and compare many popular datasets. We further propose a classification that groups the defects that others and ourselves found in such datasets.
  - The core aspect of this chapter is our proposal of a mechanisms to facilitate the creation of datasets that overcome the defects identified in our survey. Our mechanisms consist in the injection of synthetic attacks into arbitrary network traffic inputs. The injected attacks replicate the statistical properties of the network traffic inputs to disguise the injections.
  - A dataset has quality issues if it contains problems that may bias the evaluation results of systems that use such a dataset. We develop metics to determine quantitative aspects of a network dataset that reveal potential issues that degrade the quality of the dataset.
- Intrusion Detection (Chapter 4)
  - At the core of this chapter, we develop a mechanism capable of detecting distributed and collaborative attacks in large networks that can cope with massive amounts of data and requires relatively low computational resources.
  - We leverage neural network concepts, i. e., dropout and autoencoders, to build an unsupervised anomaly detection NIDSs.
  - The anomaly detection system we propose learns adequate models of normality (see Section 2.1.4) even when using training data that contains attacks.
- Community Formation (Chapter 5)
  - We develop the theory of community building within CIDSs. A community is a group of NIDS sensors that together trade off communication overhead and detection accuracy.
  - We leverage ensemble learning to create distributed models out of models that can typically only be exploited in centralized environments.

- With communities, we enable distributed CIDSs to use centralized intrusion detection mechanisms without having a SPoF.

- Intrusion Information Dissemination (Chapter 6)
  - We propose a dissemination mechanism that uses Bayesian Networks to enable CIDSs to quickly and efficiently find NIDSs observing similar events in a distributed manner. Our dissemination strategy is tailored to CIDSs.
  - Our mechanism is agnostic to the detection level: CIDSs may find distributed attacks at the *alarm level* or at our newly proposed *detection level*.
  - The mechanism we propose enables distributed aggregation and correlation that requires no centralized component.
  - By using our mechanism which leverages Bayesian Networks, members of a CIDSs may deduce information that they have not yet observed. This enables members to perform their tasks without having to wait for information to fully disseminate within a CIDSs.

- Collusion Detection (Chapter 7)
  - In this chapter, we develop a trust mechanism to detect dishonest NIDS sensors acting against the agenda of their CIDS.
  - Using clustering and GMMs, we develop an efficient mechanism that determines accurate trust scores using the reliability of sensors.
  - Our mechanism can detect one or more groups of colluding dishonest sensors. Furthermore, our mechanism successfully punishes colluders that choose to act honestly some times to fool the system.

1.6 PUBLICATIONS

Most of the work presented in this thesis has been published in peer-reviewed conferences, workshops and journals of which a considerable portion is highly ranked. Ten published works constitute the core of this thesis along with two others under review. All these publications have in common the topic of NIDSs and ML.

Finding datasets for evaluating NIDSs is a core difficulty in our field. For this reason, many of our publications touch upon the topic of creating datasets for evaluating NIDSs. Our original idea to create mechanisms to inject synthetic attacks into real traffic mimicking its properties is described in [Cordero, Vasilomanolakis, Milanov, et al., 2015]. Afterwards, we improve and expand the idea in [Vasilomanolakis, Garcia Cordero, et al., 2016]. These publications, along with another one under review, make the contents of Chapter 3.

To cope with large amounts of network traffic, we recognize the potential usefulness of an unsupervised and scalable anomaly detection NIDSs. In [Cordero, Hauke, et al., 2016], we report our findings in experimenting with Replicator Neural Networks (RNNs) to create an anomaly-based NIDS. The research and results of this publication make up Chapter 4.

There is extensive literature relating to the detection of network attacks using centralized ML algorithms. Most algorithms cannot be easily transformed to work in a distributed scenario and, therefore, were not directly usable within CIDSs. Some of our research explores adapting already existing ML algorithms to the context of CIDSs using ensemble learning. We propose a methodology to do such a task in [Cordero, Vasilomanolakis, Mühlhäuser, et al., 2015]. This methodology and our evaluation of results became the essence of Chapter 5.

We faced the problem of developing a dissemination mechanism within CIDSs when we developed an overlay-aware CIDSs in [Vasilomanolakis, Krugl, et al., 2016]. We also observe how, in related work, others take dissemination for granted and do not propose dissemination mechanisms tailored to the requirements of CIDSs. Consequently, we develop such a mechanism and present it in Chapter 6. Our mechanism is described in a publication which is currently under review.

Many ML fields, such as those involving vision or network security (e. g., [Akhtar et al., 2018; Q. Liu et al., 2018]), have recently been concerned with adversarial attacks and their mitigation. An *adversarial attack* is a set of inputs that an attacker carefully crafts to cause mistakes in the prediction of an ML model. In this same line of thought, we studied different adversarial attacks against CIDSs. Some of our research focused on probe-response attacks against CIDSs (published in [Vasilomanolakis, Stahn, et al., 2015, 2016]) and colluder resistant systems (published in [Cordero, Traverso, et al., 2018; Traverso et al., 2017]). In this thesis, we concentrate exclusively on mitigating collusion and use our publications on this matter to develop Chapter 7.

adversarial attack

We published two articles that do not directly relate to specific chapters of this thesis but, instead, are used sparsely throughout the thesis. In [Gazis et al., 2014], we propose an architecture to enable collaborators to securely share data. This work helped form some of the ideas presented in Chapter 5, Chapter 2 and this introduction. Finally, in [Vasilomanolakis, Srinivasa, Cordero, et al., 2016], we present a system for generating signatures of multi-stage attacks for misuse NIDSs. Although misuse NIDSs are not covered in this thesis, our expertise in this topic helped us to present the introduction and related work sections.

1.7 THESIS ORGANIZATION AND STRUCTURE

This section details the organizational and structural characteristics of this thesis. We begin with an explanation of how the reader should use the margin notes. We then describe how our core contributions are structured. In this description, we detail the logical order of the contributions as well as the structure each contribution follows. Finally, we give a brief outline of the thesis.

1.7.1 *Margin Notes*

This thesis uses margin notes to highlight important definitions and concepts. A margin note also points towards terminology that the reader can expect to find later in use. Margin notes should ease the process of skimming backwards through a chapter to locate a definition or term. Margin notes also identify those concepts of importance on which to focus. We recommend that the reader does not stop to read margin notes. Instead, they should be used as a reference to recall terminology or identify key concepts detailed within a section.

1.7.2 *Structure of the Contributions*

logical order of the chapters

The five core chapters and contributions of this thesis follow a logical order rather than a chronological one in relation to when we carried out research. The contributions are shown in Figure 1.2. We begin our contributions with the chapter titled *Dataset Generation*, Chapter 3, by addressing the problem of finding suitable datasets to evaluate NIDSs. With the dataset problem addressed, we move forward to the chapter titled *Intrusion Detection*, Chapter 4, to propose an unsupervised anomaly detection mechanism. Our mechanism can process the amount of data that a large network produces while still successfully detecting (relatively small) distributed attacks. However, our mechanism is centralized and does not easily translate to fully distributed scenarios. In *Community Formation*, Chapter 5, we propose concepts to build communities where centralized algorithms can be used within fully distributed CIDSs by leveraging communication overhead and detection accuracy. The *Community Formation* chapter exposes the problem that disseminating data is not easy (and left unaddressed by related work) within CIDSs. In *Information Dissemination*, Chapter 6, we create a mechanism to disseminate information that takes into account the specific needs of CIDSs. The dissemination mechanism assumes, as related work typically does, that members of a CIDSs act honestly. With the popularity of adversarial attacks, we finally take on the challenge of identifying dishonest CIDSs members using computation trust in Chapter 7.

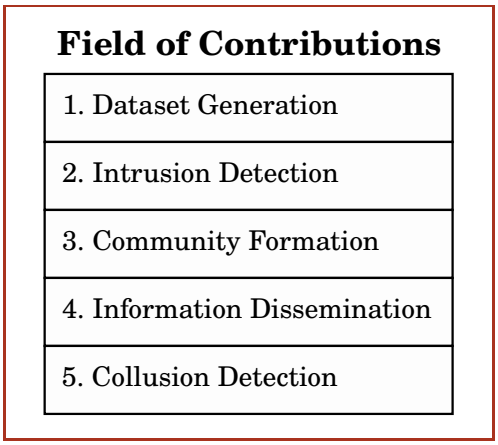


Figure 1.2: Five contributions make the core of this thesis. Each contribution builds on top or solves a basic problem introduced in the previous contribution.

context box

chapter overview

chapter-specific outline

problem statement

challenges

scientific contributions

background and related work

system section

evaluation

conclusion

Each of the five contributions that make up Chapter 3 to 7 follow the same structure. A contribution chapter starts with a box which we title *Context*. The context situates the problem tackled in each chapter with respect to the other chapters. After the context, the chapter begins with a brief description of how we solve the core problem of the chapter. We then continue with an overview of the chapter contributions. The *overview* uses Figure 1.2 and Figure 1.1 to highlight, respectively, on which chapter we are located, and on which CIDS components the chapter focuses (see Section 1.2). This is then followed by a chapter-specific *outline*.

The introduction part of each chapter begins with a small motivation. The motivation is then followed with a *problem statement*. Afterwards, we highlight the core *challenges* needed to be solved to address the problem statement. The introduction closes with the *scientific contributions* of the chapter. Each chapter deals with topics that are exclusive to itself and its contributions. Because of this, we follow the introduction with specialized *background and related work* that only applies within its respective chapter.

Chapters then follow a different structure depending on whether a system, algorithm or general novel concept is detailed. After the core details of each chapter, we close with an *evaluation* (or use case) that provides scientific insights into that which we propose. Every chapter closes with a *conclusion* that, besides summarizing the achievements of the chapter, highlight lessons learned in the process of developing the chapter.

1.7.3 *General Outline*

This thesis is structured as follows. After this introduction, we provide general background and related work in Chapter 2, where we covers topics that concern every other chapter that follows. From Chapter 3 to 7 we present the five chapters that make up the core of the thesis (as described in Section 1.7.2). Finally, the thesis closes in Chapter 8 with a general conclusion and an overlook.

2

BACKGROUND AND RELATED WORK

This chapter introduces key background topics that relate to all of the five core contributions we present in this thesis. The background topics to be presented also include related work that concerns all of our contributions. Complementary to this chapter, we cover specialized background and related work within each contribution chapter (i. e., from Chapter 3 to Chapter 7).

This chapter covers three main topics. We begin with a brief introduction to key ML concepts that we use throughout this thesis. We continue with a distilled introduction to NIDSs. Finally, we transition to describe CIDSs, which are the core topic addressed in this thesis.

2.1 MACHINE LEARNING

ML is an approach to artificial intelligence that consists in designing algorithms capable of learning without being explicitly told how to. The field is similar to the field of mathematical optimization where the goal is not only to minimize reconstruction error but also to minimize a data generalization error. In this thesis, we use several ML algorithms as a mean to improve CIDSs. Without attempting to be exhaustive or complete, this sections briefly introduces some ML concepts which are sparsely used throughout each of our contributions. Four key topics are covered. First, we describe the performance metrics that we generally use to discuss the performance of ML algorithms. Second, we explain the different data types and encodings used by ML algorithms. Third, we explain the differences between the three datasets used to train ML algorithms. Finally, we describe what anomaly detection is and how ML is used in such a task. All four topics are accompanied with resent highlights of related work.

2.1.1 *Performance Metrics*

The performance of ML algorithms is mostly measured using metrics that depend on the four prediction classes known as *true positive*, *true negative*, *false positive* and *false negative*. To explain the prediction classes, let us assume a dataset D and a data instance $d \in D$. In a scenario where data instance d belongs to one of two classes, we say that the true class of d is given by the function $c_t(d) : D \to \{0, 1\}$. The function $c_t(d)$ maps the data instance d to either class 0 or 1. Given an ML model M, we say that M predicts the class of d using the function

$c_p(d;M) : D \to \{0,1\}$. Depending on the values of $c_t(d)$ and $c_p(d;M)$ for the same d, the prediction class according to model M changes.

We illustrate the four prediction classes and how they relate to the true class ($c_t(d)$) and the class predicted by the model ($c_p(d;M)$) in Figure 2.1. The four prediction classes are defined as:

- True Positives (*TP*). $c_t(d) = 1$ and $c_p(d;M) = 1$.
- True Negatives (*TN*). $c_t(d) = 0$ and $c_p(d;M) = 0$.
- False Positive (*FP*). $c_t(d) = 0$ and $c_p(d;M) = 1$.
- False Negative (*FN*). $c_t(d) = 1$ and $c_p(d;M) = 0$.

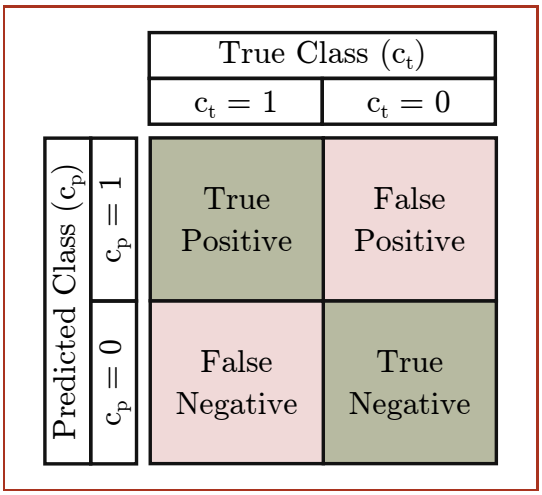


Figure 2.1: Four prediction classes are used to evaluate the predictive capability of an ML model. True Positives and True Negatives represent correctly predicted classes. On the contrary, False Positives and False Negatives signal that predictions are not correct.

Metrics combine the prediction classes in different ways to highlight diverse qualities of ML models. In the following, we describe the five most commonly used metrics to determine the performance of models. Without loss of generality, our explanations assume that data instances belong to either a *positive* or *negative* class. All metrics are applicable within a multi-class setting but are not discussed here (see [Bishop, 2006] for more information).

ACCURACY The accuracy of an ML model in a classification problem corresponds to the correct (either positive or negative) predictions made over all predictions. The metric is typically used when the classes in a dataset are balanced and is uninformative when datasets are unbalanced. Accuracy is defined as:

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN}$$

PRECISION Precision is used to measure the proportion of positive data instances that a model classified as positive. The precision metric ignores the capabilities of a model to recognize negative classes. Precision alone does not sufficiently describe the performance of a system. Therefore, the precision is often reported along the *Recall* metric. Precision is defined as:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

RECALL The recall of a model is the proportion of true positives that the model identified. As such, a model that yields no false negatives (FN) has a recall of 1.0. Precision and recall are normally inversely proportional to each other, i. e., as one is improved, the other is worsened.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

SPECIFICITY The specificity of a mode corresponds to the proportion of negative data instances correctly predicted by the model as negative. This metric is the opposite of recall and is used when the cost of incorrect negative predictions is high.

$$\text{Specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}}$$

F1 SCORE Instead of reporting multiple metrics, the *F1 Score* computes the harmonic mean of the precision and recall to obtain a single representative score. The F1 score is often the most adequate single metric for comparing different ML models.

$$\text{F1 Score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

2.1.2 *Feature Types and Encodings*

Datasets are composed of multiple data instances and each instance is made up of multiple features or attributes. Three different types of features exist. *Categorical or Nominal features* are those characterized by unsortable classes (e. g., names, gender or genre). *Ordinal features* are categorical features with a sense of order (e. g., education level, satisfaction level or age group). Features of the *numerical type* are characterized with numbers (e. g., height, number of bytes or payload size).

categorical features

ordinal features

numerical features

Many ML algorithms can be distinguished from each other according to the feature types they can process. Some algorithms can caope with categorical features while others can only use numerical ones.

Neural networks, for example, are only able to use numerical features. Many techniques exist to transform categorical or ordinal feature types into numerical types. The following list mentions some of these techniques.

- Direct Numeric Encoding. This method directly assigns a number to each possible value a feature may have. A direct encoding is often ineffective as it creates numerical representations which assign an explicit rank to values that do not have one.
- One-hot Encoding. This encoding scheme transforms one categorical feature into several binary features. The scheme consists in replacing a categorical feature with m new binary features, one for each possible value of the categorical feature. For each data instance, all m new features are set to zero except for one, set to one, that corresponds to the original value of the feature. This technique works well when the possible number of feature values is low. Otherwise, the curse of dimensionality[1] becomes a problem.
- Dummy Coding Scheme. Similarly to the One-hot encoding, this method creates $m-1$ new binary features. The m-th feature is represented by all $m-1$ features being zero.
- Feature Embeddings. This technique gives categorical attributes numerical values that associate inherent properties of the feature to notions of distance. For example, words may be assigned vectors of numbers (known as Word2Vec [Mikolov et al., 2013]) such that the following computation makes sense: $king - man + women = queen$.
- Autoencoders as Embeddings. This encoding scheme uses autoencoders to automatically find suitable numeric representations of a feature. Autoencoders are special arrangements of neural networks that reduce the dimensionality of data analogously to methodologies such as Principal Components Analysis (PCA) [B. Zhang et al., 2012]. Autoencoders find compressed representations of datasets that can be used as numerical embeddings (e. g., [Dizaji et al., 2017]). We use this technique in Chapter 4 to reduce the dimensionality of network data and create embeddings.

2.1.3 *Datasets and Model Training*

training set

The process of finding the adequate parameters (or hyper-parameters) of an ML model uses the three sets shown in Figure 2.2. The *training set* provides the data instances that a model directly uses to learn its

1 The *curse of dimensionality* is a problem ML algorithms have when learning with high-dimensional datasets. The more dimensions, the more training examples are needed to learn a model without overfitting it to the dataset.

parameters. For example, neural networks may use gradient descent on samples of the training set to find its weights and biases. The *validation set* provides the means to conduct the unbiased evaluation of a model's fit. After changing parameters, the validation set is used to determine the effects of the change. Both training and validation sets are said to be seen during training. The training set is directly seen to train a model. The validation set, however, is only indirectly seen to select better parameters (i. e., it is seen but not used during training). The *testing set* provides a global unbiased estimate of the final model. This set is not seen during training nor parameter tuning.

validation set

testing set

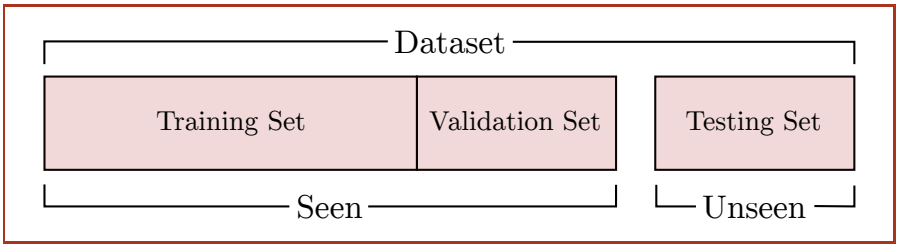


Figure 2.2: To learn the hyper-parameters of an ML model, a dataset is split into three parts. The training and validation sets are seen during training. The training dataset is directly *seen* to find parameters while the validation dataset is indirectly seen to fine-tune parameters. The testing dataset remain *unseen* until calculating the model's generalization error.

The training, validation and testing sets are assumed to be generated from the same probability distribution with parameters θ. Non-sequential datasets are further assumed to follow the independent and identically distributed (i.i.d.) principle[2]. With these two assumptions, the expected training error approximates the expected testing error as the data instances in both sets are generated from the same distribution with parameters θ [Goodfellow et al., 2016]. In ML, however, we do not know θ in advance. Therefore, the goal of an ML algorithm is to find θ by minimizing the training error and the gap between the testing and training errors. When the training error is too large, we say a model *underfits* the data. When the gap between the testing and training errors is too large, we say the model *overfits* the data.

i.i.d. principle

model underfitting

model overfitting

Preventing a model from overfitting is challenging. Although there are no techniques that can guarantee that a model does not overfit its data, we can use several techniques to assess if a model is overfitting and by how much. The family of methods known as *cross-validation* enables us to statistically determine how well a model generalizes on average (i. e., how well the model does not overfit). The technique known as *k-fold cross validation* partitions a dataset into k different subsets. The technique uses all the subsets but one to form a

cross-validation

2 In an i.i.d. dataset, all data instances are created using the same probability distribution and do not depend on each other.

training dataset. To build a validation dataset, the technique gathers the leftover subset not used for training. The training and validation datasets are then used in the normal process of training and validating a model. This is repeated k times, each time retaining different subsets of the data for validation, and the average performance of the model is computed. This final computation identifies the generalization capabilities of the model.

The performance of an ML model is often measured by its "loss" (also known as "error"). The loss is the discrepancy that exists between the prediction of a model and an expected value. The training loss of a model is the loss of the model when it is evaluated against the training set. Similarly, the validation loss is the loss of the model when it is evaluated against the validation set. During training, an ML algorithm tries to minimizing the loss of the validation set while only using the training set. Equation 2.1 shows a commonly used loss function known as the *quadratic loss*.

training loss

validation loss

quadratic loss

$$\mathcal{L}_M(D) = \frac{1}{N}\sum_{i=1}^{N}(y_M(x_i) - t_i)^2 \tag{2.1}$$

The loss $\mathcal{L}_M(D)$ is the average of the squared difference between the prediction $y_M(x)$ of model M and the known target value t, where N is the total number of data instances in dataset D. The quadratic loss function is heavily used in the ML algorithm for intrusion detection we propose in Chapter 4.

2.1.4 *Anomaly Detection*

Generally speaking, anomaly detection is the task of discovering unusual patterns in data that do not conform to pre-established notions of normality. An anomaly detection system, defined as the tuple (M, D), is composed of a normality model M and similarity measure D. The normality model M contains all that is needed to determine if some data points are normal or not. The similarity measure D is responsible for computing the distance that exists between an arbitrary data point and model M. This distance is known as *anomaly score* and is used to label something as an outlier. An anomaly score close to zero indicates that something is normal. The further an anomaly score is to zero, the more anomalous something is. When something is to be labeled as either anomalous or not, a threshold establishes the limit after which an anomaly score signals an anomaly.

normality model

similarity measure

anomaly score

Related work generally categorizes anomalies into three different classes [Chandola et al., 2009]. *Point anomalies* are data instances that are considered too far away from any other data instance (e. g., [McFowland et al., 2013]). *Contextual anomalies* are data instances that do not match the expectations brought about by a context (e. g., [X. Song et al., 2007]). *Collective anomalies* are groups of data instances that to-

point anomaly

contextual anomalies

collective anomalies

gether seem normal but are all abnormal with respect to a dataset as a whole (e. g., [Sun et al., 2006]).

The normality model M of an anomaly detection system may be created using techniques from statistics, physics or machine learning, among many other options. ML-based anomaly detection refers to anomaly detection systems that obtain their normality model M using ML. Das et al. [2007] propose to use *Bayesian Networks* to build normality models. *Rule mining* is used in the work of McFowland et al. [2013] to find rules that describe normal behavior. In [B. Zhang et al., 2012], the authors use the *subspace method* (see Section 4.2.3) to create mappings from high to low dimensionality spaces to create a normality model. Kwon et al. [2017] provide a survey on techniques based on *deep neural networks* to create normality models. As a contribution to this thesis, in Chapter 4, we propose an approach based on neural networks.

2.2 NETWORK INTRUSION DETECTION SYSTEMS

Computer networks are complex communication channels with physical and logical components that work together to transport information. When these channels were first designed halfway through the twentieth century, the security of the transported information was but a simple goal. However, with the predominance and criticality of computer networks, security has become a major concern. To protect networks, proactive and reactive approaches exist. *Proactive approaches* impose restrictions on networks to prevent intrusions. Some popular proactive approaches include the usage of firewalls, virtual private networks, security protocols or cryptography, among others. In contrast, reactive approaches monitor networks in search for indications that an intrusion is or has taken place. NIDSs fall in the category of reactive security approaches.

proactive security approaches

reactive security approaches

IDSs are systems that attempt to detect actions that compromise the integrity, availability or confidentiality of a target. Depending on the target they monitor, IDSs are grouped into two main classes. Host-based IDSs are responsible for detecting undesired activities at the device level. In contrast, NIDSs detect unwanted activities at the communication level. In this thesis, we only consider NIDSs aimed at detecting intrusions in large networks.

classes of IDSs

2.2.1 *NIDS Requirements and Difficulties*

Host-based IDSs and NIDSs share many common requirements. Butun et al. [2004] propose five different requirements that make IDSs usable in practice. One, IDSs must not introduce new weaknesses into the target they monitor. Two, IDSs need to use as few resources as possible to avoid hampering the target's functionality. Three, for the detection

five IDS requirements

to be useful, IDSs need to run continuously and transparently minimizing user intervention. Four, whenever possible, standardized protocols should be used. Finally, and most importantly, IDSs need to be reliable. A reliable IDS is one that minimizes false positives and false negatives (see Section 2.1.1 for a description of these metrics).

special NIDS requirements

NIDSs have special requirements beyond those that overlap with Host-based IDSs. While Host-based IDSs use partially structured and typically homogeneous data (e. g., system calls or event logs), NIDSs use network data. Network data has diverse properties that makes it substantially difficult and challenging to use for intrusion detection [Lakhina, Crovella, and Christophe Diot, 2004]. This is especially true when ML, the currently most used technique [Sperotto and Pras, 2010], is used as the mean to detect intrusions [Sommer et al., 2010]. To cope with the complexity, speed and size of traffic in large networks, researchers are using ML on top of network flows [B. Li et al., 2013; Soysal et al., 2010]. Section 4.2.1 gives a brief definition of network flows.

three major difficulties

Applying ML within NIDSs is not a straightforward task. In contrast to other fields, the network intrusion detection field needs to cope with three major difficulties. These difficulties are, one, the *high cost of errors*; two, the *absence of datasets*; and three, the *lack of actionable responses* against detected intrusions [Sommer et al., 2010]. These difficulties are further aggravated by the complexity and heterogeneity of network traffic. Much of today's work on NIDSs has focused on proposing solutions to these difficulties. To cope with the *high cost of errors*, for example, deep learning has been used to maximize detection accuracy while reducing false positives [Kwon et al., 2017]. In Chapter 4, we develop an NIDS that uses techniques borrowed from deep learning to also address the high cost of errors. A major issue that contributes to the *absence of datasets* is the fear of leaking confidential information, among several other problems (see Section 3.1.2). In Chapter 3 of this thesis, we propose a potential solution to this second difficulty. Finally, regarding the *lack of actionable responses*, we recognize that responding to intrusions is of such a complex nature (e. g., [Lee et al., 2002]) that it is becoming a field on its own [Anwar et al., 2017]. The term *Intrusion Response Systems* is used to refer to this field. Responding to intrusion is outside of the scope of this thesis.

first difficulty

second difficulty

third difficulty

2.2.2 *NIDS Architecture and Classification*

Figure 2.3 shows a simplified architecture of an NIDS. The architecture has four main components. The *network component* is known in more general IDS architectures as the *environment*. NIDSs only use network traffic to carry out their task. Network traffic is processed in the *sensors component* by one or many devices. In the context of NIDSs, the term "sensor" refers to an active or passive device that extract

network component

sensors component

data features from network traffic for further analysis. Some example sensors are firewalls, honeypots, routers or Network Interface Cards (NICs) acting as sniffers. The *intrusion detection module* is responsible for processing collections of network features, gathered from one or many sensors, to identify intrusions. Two main intrusion detection categories of NIDSs exist. Signature-based Network Intrusion Detection Systemss (SNIDSs) use the footprints of previously observed intrusions to detect whenever they occur again. Anomaly-based Network Intrusion Detection Systemss (ANIDSs) learn models to represent the normal behavior of network traffic. These models of normal behavior are used to detect traffic that deviates from the norm, i. e., anomalous traffic. ANIDSs are covered in more detail in Section 2.2.3. The final architectural component, known as the *response module*, is accountable for applying *actions*, or countermeasures, to a network to mitigate intrusion reported by alarms. In this thesis, the response module is not covered.

intrusion detection module

intrusion detection categories

response module

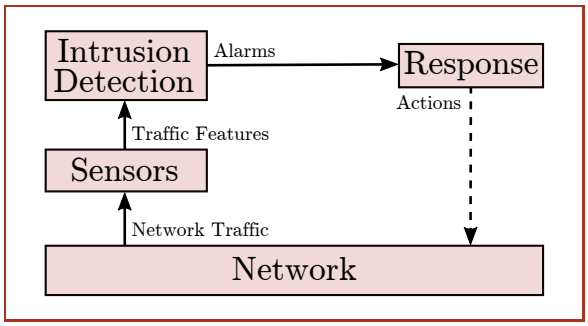


Figure 2.3: The core architectural components of an NIDS and the key data items the components transfer.

SNIDS and ANIDSs have different advantages and disadvantages. SNIDSs do not suffer from many of the difficulties discussed in the previous section (Section 2.2.1). In theory, these systems have low false positive rates and are highly effective in recognizing known attacks. To obtain signatures, instead of requiring difficult to obtain datasets, they need intrusion examples. Finally, countermeasures can be tailored to specific attacks. For these and other reasons, SNIDSs are the most widespread NIDSs (e. g., Bro [Paxson, 1999], Snort [Roesch, 1999] and Suricata[3]). Creating signatures is a major challenge of SNIDSs and many researchers have created automated mechanisms, often based on ML, to derive signatures (e. g., [Kim et al., 2004; Kreibich et al., 2004]). We have also worked in this direction [Vasilomanolakis, Srinivasa, Cordero, et al., 2016]; however, this thesis concentrates on ANIDSs as, we argue, the effectiveness of SNIDSs is in decline. SNIDSs need accurate signature databases to be effective. However, creating and maintaining these databases is becoming daunting due to

3 https://www.openinfosecfoundation.org

the constant appearance of novel threats, the popularization of encrypted communication mediums, and the growth of attack surfaces. Under these conditions, even large and constantly updated signature datasets are not sufficient to protect networks.

Despite some of their problematic aspects, ANIDSs have qualities that make them more appropriate than SNIDSs in many circumstances: ANIDSs can detect attacks that have not been seen before. For this reason, the field of NIDSs is still actively developing anomaly detection methodologies [Pimentel et al., 2014]. A popular and effective technique to identify anomalies in network traffic is to use the subspace method [Lakhina, Crovella, and Christophe Diot, 2004]. This method consists in splitting network traffic into disjoint normal and abnormal subspaces using techniques such as PCA [Ringberg et al., 2007]. Many researchers, however, have criticized PCA stating that the mechanism is not robust enough (e. g., [X. Li et al., 2006]). On the other hand, recently proposed modifications to PCA have made it more robust within networks (e. g., [Chen et al., 2016]). In Chapter 4, we propose a robust isomorphic alternative to PCA that detects network traffic anomalies.

subspace method

2.2.3 *Anomaly-based Network Intrusion Detection*

ANIDSs are based on the *suspicious hypothesis*. The suspicious hypothesis states that anomalous events are deemed suspicious from a security point of view [Estevez-Tapiador et al., 2004]. In the context of network traffic, an anomalous event refers to traffic that does not conform to the expected behavior of a network. Anomalous events are found using an anomaly detection system (M, D), as defined in Section 2.1.4. In the context of an ANIDS, the model of normality M finds representations of normal network traffic while the similarity measure D finds the distance between arbitrary networks traffic and M. If, according to D, network traffic is above a predetermined threshold, the traffic is considered anomalous. The model of normality M can be represented in many ways. Matthew V Mahoney and P. Chan [2003] use *conditional rules* to model normal behavior. *Autoencoders*, a type of neural network, are successfully used in the literature to model normality (e. g., [Dau et al., 2014]). Even models borrowed from physics, such as *Wavelets* and *Fourier transformation*, are used to create models of normality [Jiang et al., 2014].

Anomaly detection uses two independent components to construct normality models and detect intrusions. The diagram in Figure 2.4 shows a simplified example of how information flows when constructing normality models and detecting intrusions. Networks generate traffic and one or many sensors collect the traffic to extract features. Features are aggregated and then distributed to the *modeling component*. This component is responsible for learning a model M

modeling component

to represent the normality of the features it received. The learned model is shared with the *detection component* which, in turn, uses it to detect intrusions. To detect intrusions, the detection component compares the features it receives against the normality model M and, if the features are above a threshold according to similarity measure D, the traffic is labeled as anomalous. In consequence, ANIDSs need to be trained before they can detect intrusions. Note that the training process (i. e., creating a normality model) does not involve labeled network traffic and assumes that only normal network traffic is considered.

detection component

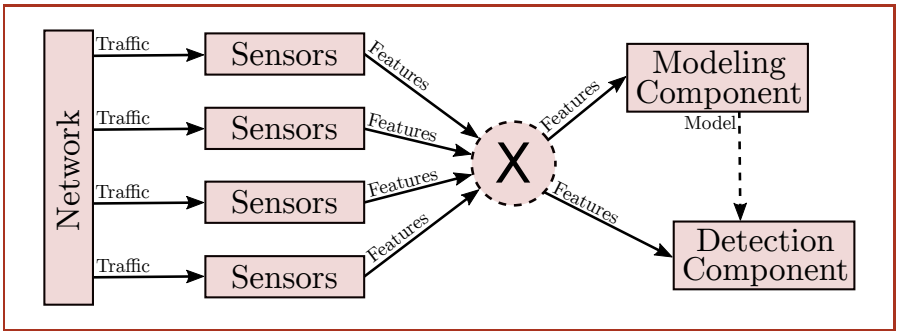


Figure 2.4: Information flow of an anomaly detection system. Network traffic is monitored by sensors that extract features. Sensors send features to a modeling component that is responsible for creating a normality model. The detection component uses the normality model to determine if some features are abnormal or not.

In principle, normality models can be constructed using arbitrary selections of features. In large networks, however, modern anomaly detection techniques use the entropy of IP header fields as features. Entropy is a metric that efficiently calculates the dispersion and concentration of a distribution [Ringberg et al., 2007]. Most network-wide intrusions affect the dispersion and concentration of IP header fields. Therefore, entropy is a suitable metric to learn normality models that represent large networks. Lakhina, Crovella, and Christiphe Diot [2004] provided most of the analysis that made the (Shannon) entropy of IP header fields a default feature in most other works. Many other researchers have also experimented and successfully demonstrated the usefulness of other types of entropies as features (e. g., the nonextensive or Tsallis entropy [Tellenbach et al., 2011; Ziviani et al., 2007]). Beyond using the entropies of IP header features, researchers have created improved anomaly detectors by mixing the entropy of other feature types beside IP header fields. A notable example is proposed by Nychis et al. [2008]. In their work, they improve anomaly detection by using the entropy of the behavior of flows (i. e., the in- and out-degree distributions of hosts).

entropy of IP header fields

The diagram in Figure 2.4 assumes that one entity is responsible for building one single model of normality. Likewise, the diagram

implies that only one entity is responsible for using the normality model in the detection component. These two assumptions limit the system in its scale. To cope with this limitation, researchers propose groups of collaborative NIDSs, or Collaborative Intrusion Detection Systems (CIDSs). These systems are the topic of discussion in the next section.

2.3 COLLABORATIVE INTRUSION DETECTION SYSTEMS

The necessity to detect collaborative attacks has brought forth collaborative defenses. Collaborative Intrusion Detection Systems (CIDSs) are collections of NIDSs that together collaborate to detect widespread intrusions. Computer networks can reach monumental sizes, creating an environment where attackers can easily conceal their activities. The goal of a CIDS is to detect those undesired activities that would otherwise be overlooked by individual NIDSs. A CIDS is composed of multiple sensors, communication channels and one or more analysis units. As in an NIDS, sensors are responsible for monitoring local network traffic. Analysis units, in contrast to an NIDS, can be plentiful and have different roles depending on whether the collaboration level of a CIDS is at the detection or alarm level (see Section 2.3.2). Analysis units share the responsibilities of the modeling and detection component of NIDSs (see Figure 2.4).

CIDS sensors

CIDS analyzers

CIDSs are complex systems that can be organized differently according to different criteria. In the coming two sections, we expand upon two different organizational paradigms of CIDSs. The first paradigm considers the communication overlay of a CIDS. The second paradigm takes into account the collaboration level at which a CIDS operates. Regardless of how they are organized, CIDSs are made up of the same components. The *architectural components* of CIDSs are presented in Section 2.3.3. This architecture plays an important role in this thesis as it is used as the foundation by which this thesis' contributions are organized.

organizational criteria

communication overlay

collaboration level

architectural components

2.3.1 *CIDS Communication Overlays*

According to the communication overlay they use, CIDSs can be organized into three different classes. Figure 2.5 shows the three communication overlays by which related work can be organized [Vasilomanolakis, Karuppayah, et al., 2015]. Centralized CIDSs tend towards the best detection accuracy given that the data of all sensors is analyzed by one single analyzer. Its obvious deficiency is that it does not scale well to large networks. Hierarchical CIDSs alleviate the scalability issue by creating hierarchies of analyzers. At each level of the hierarchy, an analyzer processes the data of a limited number of sensors. Analyzers may collect (and aggregate) the results of other analyzers to create

centralized CIDSs

hierarchical CIDSs

meta-models. The lower an analyzer is in the hierarchy, the narrower its view is and, theoretically, the less accurate its network-wide detection capabilities are. Both centralized and hierarchical overlays have SPoFs. Distributed CIDSs, in contrast, do not have a SPoF. They rely on having many analyzers that may also act as sensors. Analyzers establish collaboration by regularly sharing information with all others. In doing so, detecting network-wide intrusions is possible at the cost of high communication overhead.

distributed CIDSs

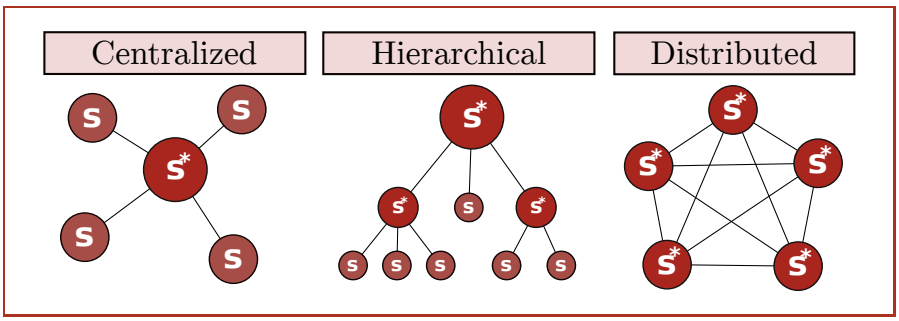


Figure 2.5: Examples of the three different communication overlay classes of a CIDS. A centralized CIDS has one analyzing node ($s^\star$) towards which many sensors (s) connect. A hierarchical CIDS has stacked analyzers and sensors connected in a hierarchical way. A distributed CIDS consists of many analyzers connected together.

Many examples of CIDSs exist that use different communication overlays. The first CIDSs were mostly centralized. Today, modern CIDSs tend to be distributed and focus on improving their performance in comparison to centralized CIDSs. Some classic centralized CIDSs are proposed in [Cuppens et al., 2002; Kannadiga et al., 2005]. In these classic systems, a central unit processes all network traffic. In the are of hierarchical CIDSs, Z. Zhang et al. [2001] propose one of the first systems of this type. They built a system that would preprocess, correlate, and aggregate sensor data until the data converged onto a root analyzer. One of the challenges of distributed CIDSs is to employ adequate dissemination mechanisms. C V Zhou et al. [2007], for example, use a publish-subscribe overlay on top of a Distributed Hash Table (DHT) and Peer to Peer (P2P) network. Due to the effectiveness of P2P networks, others have used it to propose alternative systems (e. g., [Z. Li et al., 2006; Marchetti et al., 2009]). In Section 6.1, as the fourth contribution of this thesis, we propose a distribution mechanism that improves upon the aforementioned proposals and the state of the art.

2.3.2 *CIDS Collaboration Levels*

We propose a new organizational paradigm that categorizes CIDSs depending on the level at which CIDS members collaborate. We rec-

ognize two distinct levels of collaboration according to how data is transferred between the *sensor module* and *intrusion detection module* shown in Figure 2.3. In the case of an NIDS (as illustrated in the figure), the modules always exchange data at the feature level. Sensors are responsible for extracting relevant network features. These features are used by the intrusion module to learn and detect intrusions. Analogously, CIDSs also have sensors and analysis units; however, they may interact at a detection level (i. e., feature level) or alarm level. At the *detection level*, sensors and analyzers exchange features to build models that describe network-wide behaviors. At the *alarm level*, sensors exchange features only with few analyzer. Each analyzer is then responsible for independently building their own intrusion detection model. Analyzers establish collaboration with others by exchanging, aggregating and correlating the alarms they generate with their own models.

detection level
alarm level

ALARM LEVEL COLLABORATION Almost all CIDSs operate at the alarm level [Vasilomanolakis, Karuppayah, et al., 2015]. Notable classical examples of these type of systems include the work of Dash et al. [2006]. They propose an anomaly, distributed and network-based CIDSs that can detect threats that slowly propagate through large networks. Like this work, many others focus on general solutions that apply within any network (other notable examples are, e. g., [Garcia et al., 2004a; Q. Zhang et al., 2003]). The current trend, however, focuses on proposing CIDSs at the alarm level that apply within networks of a specific domain. In the works of Gamer [2012] and Marchetti et al. [2009], CIDSs are created for the purpose of detecting intrusions in backbone networks. In [Sedjelmaci et al., 2015], CIDSs are applied within vehicular networks. The popular domain of smart grids have seen CIDSs proposals tailored to the domain (e. g., [Hong et al., 2017; Xiaoxue Liu et al., 2015]).

DETECTION LEVEL COLLABORATION At the detection level, almost no CIDSs have been proposed. However, we argue that the field of distributed ML model learning is isomorphic to the field of CIDSs that operate at the detection level: In both fields, the goal is to distributedly create ML models that minimize communication overhead while maximizing the accuracy of one or more models. Many distributed ML algorithms are somehow based on ensemble learning [Peteiro-Barral et al., 2013]. Ensemble learning is the technique of combining multiple models to make decision (see Section 5.2.2 for more details). Model combinations can be made by mixing the predictions of the individual models or by mixing the models themselves [P. K. Chan et al., 1993]. Both model combinations are suitable within CIDSs: analyzers can either create and share their own models or share only significant statistics [Lazarevic, Nisheeth Srivastava, et al., 2009]

to calculate one single global model (at each analyzer). In Chapter 5, we propose a CIDS at the detection level that mixes model predictions to increase detection accuracy.

2.3.3 *CIDS Architectural Components*

CIDSs are complex systems composed of multiple subsystems interacting together with the goal of detecting network-wide intrusions. In order to better study CIDSs, Vasilomanolakis, Karuppayah, et al. [2015] propose a CIDS architecture composed of five building components. The architecture is shown in Figure 2.6. The design principle of the architecture is that of *separation of concerns*. Each component operates independently of each other, only requiring the services of those other components with which it is in contact. In the reminder of this section, we describe the responsibilities and mention the noteworthy related work that relates to each component. In our descriptions, we exclusively concentrate on CIDSs that operate at the *network level*, i. e., they only monitor network traffic in search for intrusions. We disregard CIDSs operating at the *host level* as they are outside the scope of this thesis.

five building components

separation of concerns

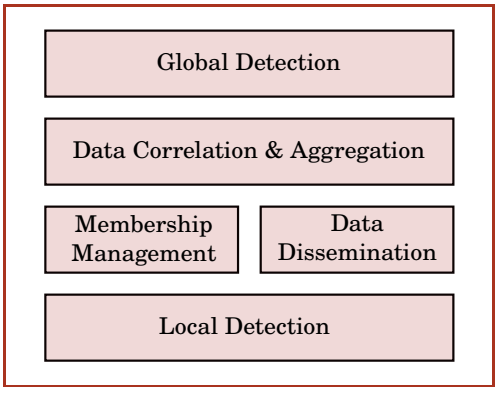


Figure 2.6: The architecture of CIDSs. Five components interact together to create a collaborative environment. Analyzers operate at the *local detection* level. Though *membership management* and the *dissemination of data*, they establish overlays for communication. Shared data is *correlated and aggregated* before it is used to build a *global detection* component.

An overview of the architecture and data flow is as follows. At the *local detection* component, sensors and analyzers are responsible for monitoring network traffic and generating local data. Local data is then disseminated to other analyzers with the help of the *data dissemination* component. This component uses the services provided by the *membership management* component to ensure that all interested parties receive the local data. The *data correlation & aggregation* component provides the means by which analyzers efficiently collect data.

architecture overview

The collected data is used by the *global detection* component to provide intrusion detection mechanisms at a network-wide scale.

2.3.3.1 *Local Detection Component*

The *local detection* component is responsible for capturing and processing network traffic to provide high-level information to other components. Local detection components have two different modes of operation depending on whether a CIDS operates at the alarm or detection level. At the alarm level, the local detection component creates its own signature or anomaly-based models which it uses to detect local intrusions. Alarms, which are the results of successfully detecting intrusions, are forwarded to other CIDS members. At the detection level, analyzers share features with each other to build distributed ML models.

operating at the alarm level

operating at the detection level

Signature and anomaly-based NIDSs can be used as a local detection component. The most popular signature-based NIDSs include Bro [Paxson, 1999], Snort [Roesch, 1999] and Suricata[4]. Anomaly-based NIDSs are scarce and often custom built. Some well known software systems that include anomaly detection capabilities are Hogzilla[5] and CFEngine[6]. In the second contribution of this thesis, found in Chapter 4, we propose an anomaly detection mechanism that can operate within this component.

2.3.3.2 *Membership Management Component*

The role of this architectural component is to establish communication overlays between CIDS members. Communication overlays can be classified as static or dynamic. A static overlay is one where each member manually specifies fixed connections to other members. This overlay class is the default used one by centralized systems where members know the centralized component with whom to connect. In contrast, dynamic overlays are established and reorganized when appropriate. In such overlays, members do not have fixed connections. Instead, connections are established as needed according to a protocol or algorithm. Distributed CIDSs make use of dynamic overlays to accommodate a variable number of members and to prevent a SPoF.

The membership management component may use a diverse set of technologies to establish dynamic communication overlays for CIDS members. The most basic of this technologies is known as P2P networks [Marchetti et al., 2009]. This technology relies on members of a P2P network having the ability to store lists of neighbors (i. e., other systems to whom data can be sent), and the ability to populate a neighbor list by asking other members for their neighbors.

4 https://suricata-ids.org/
5 https://ids-hogzilla.org/
6 https://cfengine.com/

DHTs, for instance, enable collaborating members to distributedly save and share information using structured data routing [Androutsellis-Theotokis et al., 2004].

Membership management is not a solved and obvious task; CIDSs need to solve diverse challenges to create effective communication overlays. Information shared between the members of a CIDSs is often of a sensitive nature. Whether members share alerts or features to build models, members may choose to restrict others from directly observing what they are sharing. Therefore, the dissemination strategy, as taken care by the *data dissemination component*, influences how communication overlays are established. For example, the dissemination strategy we propose in Chapter 6 preserves a degree of data privacy and therefore relaxes the concerns that members may have when overlays are built. Membership management is also influenced by the trust that exists between the members of a CIDS. We propose a collusion detection mechanism in Chapter 7 which may be used to restrict how overlay links are established depending on how much CIDS members trust each other. Overlay links may be established randomly or with a goal in mind. We propose a membership management mechanism in Chapter 5 that establishes links between members with the goal of creating communities.

2.3.3.3 *Data Dissemination Component*

While the membership management component establishes communication channels, the *data dissemination* component is responsible for distributing data through those channels. Centralized CIDSs use simple dissemination mechanisms where all data is forwarded to one central component. Hierarchical CIDSs distribute data in one direction, bottom-top, using well established communication links. Distributed CIDSs, however, require more sophisticated mechanisms as communication links may be dynamically established and removed.

Dynamic overlays can use many distribution techniques, each having its own advantage and disadvantages. *Flooding* a network is the most basic technique that can be used to distribute data in a communication overlay. In its naïve implementation, flooding consists in letting each participant directly transmit data to its neighbor. Upon receiving data, a neighbor forwards the data to every other neighbor from which the data did not originate. This approach has the obvious disadvantage of having a high communication overhead. However, it guarantees that every participant sees everything that is shared. Alternative mechanisms include *gossiping*. Gossiping is similar to flooding but, instead of sending data to every neighbor, data is selectively transmitted to others following a probability distribution [Kermarrec et al., 2007]. This mechanism drastically reduces the communication overhead at the price of not guaranteeing that every participant receives all data. Gossiping, along with many other similar techniques,

flooding

gossiping

is known as *epidemic routing* [Kostić et al., 2003]. Other viable dissemination techniques include *publish-subscribe* [Daubert et al., 2016] and *multi-cast routing* [Bye et al., 2010]. In Chapter 6, we propose a dissemination mechanism customized for CIDSs that minimizes communication overhead and provides deduction capabilities to CIDS members.

publish-subscribe

multi-cast routing

Dissemination strategies often involve some form of compressed data representation. The family of data structures known as Probabilistic Data Structure (PDS) are able to store some data property in sub-linear space. This means that PDSs use less memory than the number of properties they track. *Bloom filters* [Locasto et al., 2005], for example, store whether something been observed or not. *Sketches* store how many times something has been observed. We use this type of data structures in Section 6.1 to create a data dissemination mechanism. We refer the reader to the specialized background in Section 6.2 for more details about PDSs.

2.3.3.4 *Data Correlation and Aggregation Component*

The terms correlation and aggregation in CIDSs have a special meaning. Generally speaking, *correlation* is the process of finding statistical dependencies between random variables. *Aggregation* is the process of summarizing similar data items together into a single representative item. Within the context of CIDSs, the meaning of these two terms change depending on the CIDS's collaboration level (see Section 2.3.2). At the alarm level, CIDSs correlate alarms to find groups of alarms that show signs of a dependent relation. Aggregation at this level implies mixing similar alarms together to create one alarm. The output at this level consists of meta-alarms (i. e., alarms of alarms). At the detection level, CIDSs correlate features to do feature selection for ML; and aggregate features into summary statistics. The output at this level are collections of data features.

correlation

aggregation

alarm level output

detection level output

According to the intended outcome, correlation at the alarm level can be grouped into four distinct categories. *Attack-based* correlation aims at identifying a group of alarms that follow after specif attacks (e. g., [Garcia et al., 2004b]). *Similarity-based* correlation finds similar alarms based on predefined distance metrics (e. g., [Eckmann et al., 2002]). *Multistage-based* correlation attempts to find the alerts that are generated by attacks that involve multiple steps (e. g., [Vasilomanolakis, Srinivasa, and Mühlhäuser, 2015]).

attack-based correlation

similarity-based correlation

At the detection level, exchanged features can be aggregated using sufficient statistics [Caragea et al., 2004]. Sufficient statistics are those statistics that alone can be used to generate an ML model. More formally, most ML algorithms use some statistics computed from a dataset D to generate their model M. A statistic $s(D)$ of dataset D is sufficient when it is enough to compute the parameters θ of model M. For example, the sufficient statistics of a Gaussian model are the mean and standard deviation of D. Within our contribution presented in

Chapter 6, we develop a mechanism that distributes sufficient statistics (i. e., feature distributions).

2.3.3.5 *Global Detection Component*

This architectural component uses the information provided by the aggregation and correlation component to provide network-wide intrusion detection capabilities. When a CIDS operates at the alarm level, this component finds network-wide attacks by looking at the aggregated and correlated alarms issued by the local detection component. Besides locating network-wide attacks, at this level of operation, the global detection component is responsible for filtering and ranking alarms by their level of importance. Widespread and repetitive alarms are ranked higher as it is assumed that such alarms have a higher impact at a network-wide level. At the detection level , this component is responsible for turning data features into ML models. When functioning at a detection level, CIDSs share ML models (or the means to create unified ML models) with all its members. The members can then use these models to detect network-wide attacks.

global detection at the alarm level

global detection at the detection level

3

DATASET GENERATION

CONTEXT

This chapter constitutes the first major contribution of this thesis. In the following sections, we present research that leads us to derive the concepts and theory needed to blend synthetic attacks into network traffic captures. Our concepts are realized and demonstrated in the form of a toolkit. This toolkit facilitates the creation of useful datasets for the filed of NIDSs by creating and blending synthetic attacks into arbitrary network traffic. The datasets we can generate improve upon commonly used public ones, many of which are known to contain significant problems [Creech et al., 2013; Matthew V Mahoney and P. K. Chan, 2003].

The following sections also lay down the foundations on which some of the subsequent chapters are evaluated. With our toolkit, termed Intrusion Detection Dataset Toolkit (ID2T), we create labeled datasets from unlabeled network traffic to detect, for instance, attacks in a backbone network or members colluding within a CIDS.

DATASETS are one element among many that belong to the essential means needed to develop proper and empirically tested Network Intrusion Detection Systems (NIDSs). In this chapter, we review the publicly available datasets most commonly used in the network security field. In our review, we compare the capabilities of each dataset and present a classification of their defects. To alleviate most defects and further enable researchers to develop their own datasets, we develop an Intrusion Detection Dataset Toolkit (ID2T).

An overview of this chapter is shown in Figure 3.1. On the left side, the figure shows where this contribution lies in relation to the others. Our contributions begin with the introduction of methods and means to generate useful datasets for evaluating NIDS. This ability to create useful datasets is exploited in subsequent chapters to evaluate our research. As shown to the right of Figure 3.1, this chapter is related to the bottom and top layer of the CIDS architecture we reference (see Section 2.3.3). Our dataset generation mechanisms enable us to adequately test local and global detection mechanisms. This is achieved by injecting known synthetic attacks into network traces; thereby, creating labeled datasets. These labeled datasets are used to evaluate the sensitivity of local or global detection mechanisms to attacks of different nature.

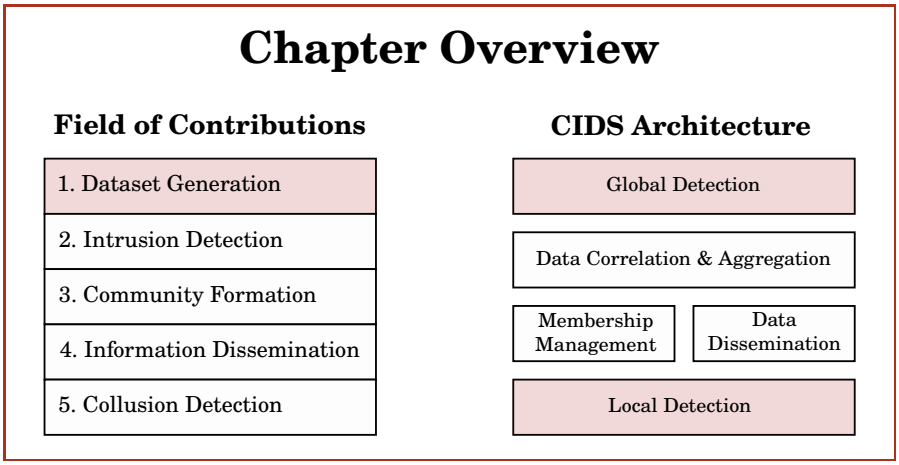


Figure 3.1: This chapter constitutes the first contribution of this thesis: *Dataset Generation*. This contribution is tied to the highlighted layers of our referenced CIDS architecture: *Global Detection*, and *Local Detection*.

This chapter is structured as follows. The chapter begins by introducing and motivating the demand and the problems of generating datasets through the injection of synthetic attacks. The introduction also includes our requirements for tools that create synthetic attacks as well as requirements for datasets to be useful in the CIDS field (Section 3.2). In this field, datasets play a major role. It is for this reason that we summarize the major datasets and tools that create datasets through the injection of synthetic attacks. Our dataset summary also includes a defect analysis that categorizes common problems found in real or synthetic datasets (Section 3.3). ID2T, the toolkit that we propose to generate datasets, is then introduced along with a full description of its components (Section 3.4). Within the architecture of ID2T, the TIDED and *Attacks* modules stand out above the others. The section of the TIDED module explains the metrics used to detect potential issues in the datasets created by ID2T or any other dataset (Section 3.5). The section of the *Attacks* module describes the different attacks ID2T can inject to create labeled datasets (Section 3.6). We present two use cases to demonstrate how ID2T can be used to generate datasets for the evaluation, replication and comparison of NIDSs (Section 3.7). Finally, we conclude the chapter with some lessons learned, future work and a chapter summary (Section 3.8).

3.1 INTRODUCTION

The threat landscape of network communications is constantly pushing researchers and network operators to develop and deploy more elaborate and capable Network Intrusion Detection Systems (NIDSs). Due to the lack of modern standardized datasets, security practitioners cannot easily determine and compare the capabilities of different

NIDSs under similar conditions. Furthermore, those public and frequently used datasets lack the attack diversity needed to test modern systems.

Today, it is difficult to obtain or produce reliable datasets that can be used to accurately test and evaluate up-to-date NIDSs. Furthermore, most publicly available datasets contain defects or restrictions that limit their usefulness [Koch et al., 2014]. Network communication paradigms evolve rapidly and NIDSs need to keep up, yet datasets are not updated and continue to reflect network states that are no longer relevant. For example, the widely used DARPA 1999 dataset [Lippmann et al., 1999] was generated using an outdated version of the TCP protocol [Matthew V Mahoney and P. K. Chan, 2003]. Using outdated datasets to evaluate NIDSs may lead to conclusions that do not adequately apply to modern networks. To overcome the challenge to properly evaluate systems that generalize well to modern networks, we require datasets that conform to certain quality standards.

3.1.1 *Problem Statement*

We wish to tackle the task of generating reproducible and modern datasets that are useful for testing, comparing and evaluating NIDSs. We do not focus, however, on creating a single dataset that will eventually become outdated. Instead, we develop the theory and concepts needed to inject synthetically generated malicious network traffic into arbitrary traffic. We realize the theory and concepts in the form of a toolkit. Our toolkit, known as the Intrusion Detection Dataset Toolkit (ID2T), replicates the properties of network traffic, when appropriate, into synthetically generated malicious traffic. Through this scheme of replicating properties, ID2T blends synthetic traffic with real traffic, creating labeled datasets that disguise synthetic traffic. The disguising process reduces potential sources of dataset defects and bias.

PCAP files

Most datasets in the field of NIDSs are distributed as Packet Capture (PCAP) files. PCAP files contain records of network packets where each packet, for our convenience, can be considered as originating from either an unknown or a malicious source. We term the packets that originate from unknown sources *normal or background traffic*. Conversely, we denominate those packets originating from malicious sources as *attack or malicious traffic*.

background traffic

malicious traffic

Figure 3.2 shows the basic inputs and outputs of ID2T. ID2T takes a PCAP file as input and used it as background traffic. ID2T uses the properties of background traffic to create synthetic attacks and inject them into the background traffic. When we supply ID2T with background traffic free of attacks, ID2T creates a dataset with labels that functions as the ground truth. As background traffic free of attacks is often difficult to acquire, ID2T facilitates the discovery of anomalous

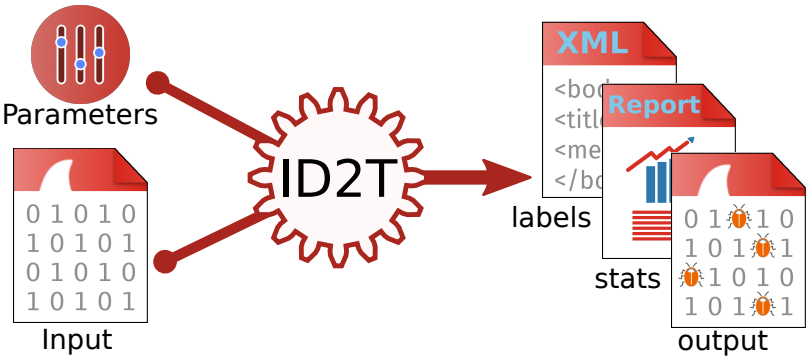


Figure 3.2: As inputs, ID2T takes a PCAP file that is used as background traffic and a set of parameters that define attacks to inject and their specifications. As outputs, ID2T creates a new PCAP file with the background traffic injected with synthetic attacks, a collection of statistics related to the background traffic, and a file containing the labels of each injected attack.

or exceptional patterns in the background traffic. The user-supplied parameters specify attacks and their properties.

ID2T generates three outputs. The first output is a PCAP file that simulates that (disguised synthetic) attacks took place at the same time as when the background traffic was recorded. The second output is a collection of statistics that relate to the input background traffic (e. g., packet rates, Time to Live (TTL) distribution or IP source entropy). The third output is a file with labels that detail when an attack takes place and its characteristics.

3.1.2 *The Challenges of Creating Adequate Datasets*

Datasets are some of the most important means to evaluate NIDSs. With ID2T, we propose one potential solution to the long-standing issue of datasets that are outdated, unfit, unavailable or cannot be easily replicated in the field of NIDS. With our solution, we attempt to address six major challenges we identify as responsible for the lack of modern datasets in the field. These challenges are listed below.

1 Privacy Concerns. Network data is prone to contain sensitive information in the payloads of network packets. Many datasets are not published due to the fear of leaking sensitive information. With ID2T, there is no need to share background network traffic. Instead, only the attack generation process is shared and the user is responsible for providing background network traffic (which is generally easy to collect or obtain).

2 Safety Concerns. Manipulating malicious programs (e. g., viruses, malware or botnets) to generate malicious network traffic has the potential of compromising or affecting unsuspecting systems. Sandboxes and virtual environments may reduce the risk of undesired

security incidents. Nonetheless, it is generally undesirable to replicate or update datasets that depend on the manipulation of malicious programs. ID2T attempts to programmatically replicate the effects of network attacks without incurring in dangerous activities.

3 Label Availability. Datasets are most useful when they contain accurate labels that detail the attacks or issues they contain. Labeling network data, however, is a difficult, tedious and vague task: label-worthy events consist of multiple packets, many of which, by themselves, would not necessarily signal malicious intent. ID2T automatically creates labels for any injected synthetic attack. Furthermore, ID2T can to mark all packets associated with an injected attack without human intervention.

4 Renewability Problems. Due to how fast network communications evolve [*Worldwide Infrastructure Security Report* 2014], datasets need regular updates to reflect the latest attacks and network paradigm changes. Otherwise, datasets risk becoming quickly outdated. With ID2T, instead of releasing new updated datasets, only attacks need to be programmed into the toolkit. The Application Programming Interface (API) that ID2T provides facilitates this task.

5 Flexibility Shortfall. Most datasets are created using fixed network configurations with attacks targeting systems within the network. Due to fixed network configuration, these datasets suffer the risk of becoming unfit or outdated. Lacking alternative network configurations means that, as systems specialize in detecting attacks in the fixed network, their generalization capabilities may diminish. As ID2T injects synthetic attacks into arbitrary network configurations, attacks can be simulated to occur in any provided network configuration.

6 Payload Availability. Because of privacy reasons, technological limitations, modeling difficulties or monitoring restrictions, packet payloads are often not provided in datasets. In this circumstances, NIDSs that use information at the packet level are automatically left out. ID2T models and includes the payload of the attacks it injects.

To overcome these challenges, we have established two sets of requirements. The first set relates to the inherent properties of datasets. The second set relates to the process of creating and injecting synthetic attacks that replicate network properties. These requirements are further divided into functional and non-functional and detailed in Section 3.2.

3.1.3 *Chapter Contributions*

This chapter is constituted by three main contributions that target the development and sharing of datasets suitable in the field of NIDSs.

In the first contribution, we present a comprehensive survey of commonly used datasets. We analyze the datasets and derive a classification of common dataset defects. In the second contribution, we develop a modular toolkit, termed ID2T, that facilitates the creation of datasets through the injection of synthetic attacks. Our toolkit takes into account our defect classification and actively tries to mitigate defects. Finally, in our third contribution, in a module of ID2T termed TIDED, we develop metrics to analyze PCAP files. The metrics are used to determine if the PCAP files contain suitable quantitative characteristics to act as background data for a dataset. To demonstrate the usefulness of ID2T, we present two use-cases that demonstrate how the toolkit can be used to reproduce the evaluation of NIDSs by creating our own datasets.

3.2 REQUIREMENTS OF DATASETS AND INJECTION TOOLS

We identify eight requirements that relate to the datasets needed in the NIDS field. On the one hand, there are requirements that apply to the datasets themselves. On the other hand, requirements apply to the generation and injection of synthetic attacks. All requirements are derived from related work. Additionally, our defect analysis presented in Section 3.3 adds to the requirements of datasets for NIDSs. The development of ID2T adds to the requirements of generating and injecting synthetic attacks. We classify all requirements into functional and non-functional ones.

3.2.1 *Requirements of Datasets Suitable in the Field*

We derive the following requirements from surveys [Vasilomanolakis, Karuppayah, et al., 2015], related work [Bhuyan et al., 2015; Koch et al., 2014; Matthew V Mahoney and P. Chan, 2003; Shiravi et al., 2012] and our experience in the field of IDSs [Cordero, Vasilomanolakis, Milanov, et al., 2015; Vasilomanolakis, Garcia Cordero, et al., 2016; Vasilomanolakis, Krugl, et al., 2016]. The following requirements are targeted at making datasets suitable to testing, evaluating and comparing NIDSs.

- Functional Requirements
  1. Payload Availability. Datasets should include packet payloads so that they are useful to test all types of NIDSs. Many NIDSs rely on deep packet inspection to recognize intrusions.
  2. Labeled Attacks. Datasets must be labeled such that the individual packets of an attack are associated to a label.
  3. Ground Truth. Dataset labels must be accurate. This implies that there is no benign traffic labeled as an attack and that there are

no attacks without an associated label. Without ground truth, we cannot compare different NIDSs with conclusive results.

4 Renewable. Datasets must be updated to include new attacks and emergent network patterns. With the speed network protocols and patterns change, a dataset can only remain relevant if it is continuously renewed.

5 Flexible. Datasets must have the ability to test different scenarios with distinct scopes (e. g., anomaly or signature-based detection in office, house or backbone network environments).

- Non-Functional Requirements

1 Public Availability or Replicability. Datasets should be public or, if not possible, easy to replicate. Many datasets, although originally public, are now difficult to obtain due to lack of maintenance from the side of the creator. Releasing a dataset to the public is therefore desired.

2 Interoperability. Datasets should be shared using a common and widespread format. The most popular file format for sharing network packet captures is the PCAP file format.

3 Quality. Datasets need to actively minimize or avoid creating issues and defects. Quality datasets reduce the bias that may exist when evaluating the capabilities of NIDSs. We distinguish between issues that arise from the generation of real traffic, and from the creation of synthetic traffic. In Section 3.3, we detail and categorize issues found in commonly used datasets.

3.2.2 *Requirements for Creating Synthetic Traffic*

Tools need to take into account a series of requirements to facilitate the task of injecting synthetic traffic. The following functional requirements are derived from observations of how datasets are compiled (e. g., [Shiravi et al., 2012]) and how other tools create synthetic traffic (e. g., [Brauckhoff et al., 2008]). The non-functional requirements are targeted at tools capable of processing arbitrary PCAP files.

- Functional Requirements

1 Packet-level Injection. Synthetic attacks need to be modeled at the packet level (instead of flow level, for example) which is the lowest common denominator for most NIDSs.

2 PCAP Interoperability. Synthetic attacks need to be injected into arbitrary PCAP files. To do this, attacks need to adjust their properties to match the statistical properties of the traffic in the PCAP.

3 Minimal User Interaction. Synthetic attacks should be created without much user interaction. A user should be expected to

only specify an initial set of parameters that are enough to generate attacks. The generation process needs to be deterministic so as to enable others to replicate a set of attack injections.

4 Packet-volume or Payload-based Attacks. Synthetic attacks should be created to model either packet or payload-based attacks. Attacks such as exploits rely on the ability to model the payloads of packets while DDoS attacks need to alter the number of created packets.

- Non-Functional Requirements
  1 Scalability. Injection tools need to process capture files of large networks. Therefore, the parsing and collection of statistics needs to be efficient.
  2 Extensibility. Injection tools need to be easily extended with new attacks to cope with the evolving threat landscape.
  3 Usability. Injection tools should not require specialized hardware to create synthetic traffic. At the same time, users should be involved as little as possible in the injection process.
  4 Open Source and Public Availability. Injection tools, to be useful to the community, need to be public and open source. Many tools and datasets become inaccessible or are lost because their creators stop maintaining them.

3.3 RELATED WORK AND DEFECT ANALYSIS

In the NIDS field, researchers are constantly looking for better datasets to evaluate, compare or replicate the results of others [Abt et al., 2013]. This need is especially evident, for example, when developing anomaly-based NIDSs, where supervised methods that need labeled data dominate [Abt et al., 2013]. In this section, we review commonly used static datasets in the field as well as tools that dynamically generate datasets. We then derive and propose a classification of common dataset defects.

In Figure 3.3, we use a timeline to show the publication year of the 18 static datasets and tools we analyze. An arrow to the right of a dataset name indicates that the dataset has been updated after it was first published.

3.3.1 *Static Datasets*

We define static datasets as those generated either from real or synthetic traffic that, once created, are not altered. Many such datasets exist, each with different degrees of popularity and deficiencies. In the following, we briefly describe each static dataset. Subsequently, we present a summary and comparison table of all datasets.

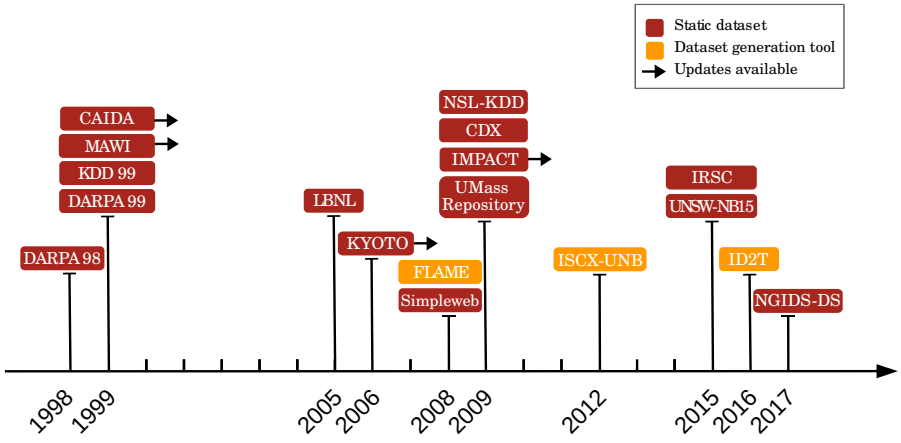


Figure 3.3: Timeline of the published year of static datasets and dataset generation tools. We show static datasets in red blocks and dataset generation tools in yellow blocks. An arrow to the right of a block indicates that updates exist for the dataset.

DARPA 98 AND 99 Created by the Lincoln Laboratory of the MIT to enable an offline intrusion detection evaluation challenge set by DARPA, these datasets are still widely used in spite of their age [Abt et al., 2013]. The datasets consists of records of simulated traffic (of hundreds of users) between a United States Air Force base and the Internet. It was a first attempt at creating an "objective, repeatable and realistic [dataset]" [Lippmann et al., 1999]. The dataset contains different threats that range from network scans to privilege escalation exploits. A total of five weeks of data are provided: two weeks of normal traffic, one of labeled attack traffic and one of unlabeled attack traffic. The datasets have been found to contain many defects that may bias the detection capabilities of anomaly detectors [Matthew V Mahoney and P. K. Chan, 2003]. For example, as a notable defect, all network packets that belong to an attack have a fixed TTL value.

KDD 99 The KDD 99 dataset was created in 1999 for a competition aimed at developing NIDS [KDD Cup 99, 1999]. The dataset is a collection of records of network flows extracted from the Defense Advanced Research Projects Agency (DARPA) 98 dataset. Each record contains 31 features. With such a dataset, developers do not need to engineer features and can fully concentrate on developing NIDSs. The KDD 99 dataset is still used today despite it being outdated and having well known issues [Creech et al., 2013].

MAWI The MAWI dataset [Fontugne et al., 2010] is a collection of PCAP files containing backbone network traffic flowing between Japan and United States. The dataset exists since 1999 and, as of today, is updated almost every day with new PCAP files. Each PCAP contains 15 minutes of anonymized traffic and, due to the amount of recorder

traffic, may be up to 20GiB in size. Despite the advantage of its recentness, the dataset has other limitations that hinder its usage in the field: packet payloads are not available, IP addresses are inconsistently scrambled among different PCAPs, and ground truth is not available. Anomaly detectors are used to label the PCAPs in an attempt to leverage the inherent problems of labeling real network traffic.

CAIDA The organization known as the Center for Applied Internet Data Analysis (CAIDA) [CAIDA, 2018] gathers, anonymizes and publishes network traffic captures of different purposes. Six datasets are available for the specific purpose of assisting security researchers, each with specialized attacks or anomalies. However, the datasets are limited in scope and have no labels.

LBNL This dataset, created by the Lawrence Berkeley National Laboratory (LBNL) and Berkeley's International Computer Science Institute (ICSI), contains around 100 hours of traffic activities recorded from a large enterprise network. With a size of 11GiB, the traffic consists of background network traffic with no known attacks. The dataset has been used to analyze large networks [Nechaev et al., 2010], but has a limited scope in the context of NIDSs. The dataset is now outdated (having been collected in January 2005) and does not contain labels.

KYOTO The Kyoto dataset [J. Song et al., 2006], rather than being a collection of network captures, consists of a series of records of features extracted from traffic captured by honeypots. The honeypots are deployed in the University of Kyoto and implement advanced mechanisms of disguise [J. Song et al., 2008]. The dataset has been updated up to December 2015. The records use the same 14 features as those used in the KDD 99 dataset plus 10 additional features related to the output of diverse IDSs. Ground truth is not available and the original packet captures are not provided.

SIMPLEWEB In an attempt to publish open datasets to evaluate and compare NIDSs, the Simpleweb/University of Twente dataset was created [Barbosa et al., 2010]. The dataset consists of eight *traces*, with varying formats, of network traffic captured in a university network. All traces have been anonymized and, if applicable, stripped of their payloads. One trace, from the eight available, consists of features extracted from network traffic that passed through a honeypot [Vasilomanolakis, Karuppayah, et al., 2015]. All traffic from this trace is considered (and labeled) as malicious [Sperotto, Sadre, et al., 2009]. The other traces do not contain labels.

UMASS REPOSITORY The Laboratory for Advanced System Software from the University of Massachusetts has a public repository of, among many things, heterogeneous network traces [Liberatore et al., 2018]. The traces are collected from specialized scenarios, reflecting specific network configurations, threats, situations and attacks. The format and labels of the traces vary. Most of them are synthetically generated and the ground truth may or may not be available.

IMPACT The community known as the Information Marketplace for Policy and Analysis of Cyber-risk & Trust (IMPACT) provides diverse datasets related to cyber-security [IMPACT, 2017]. The datasets are provided by several partners and do not conform to particular formats. Recent and old datasets exist with different combinations of labeled attacks or synthetic traffic. Access to the datasets is restricted to researchers from the USA and other authorized countries.

CDX Created in 2009 from the Cyber Defense Exercises (CDXs) competition [Sangster et al., 2009], the CDX dataset provides labeled traces of network traffic. All traffic originating from participants in charge of compromising other systems is labeled as malicious. All other traffic is considered normal or benign. Although the captured traffic consists of network traces of human activities, it does not conform to the expectations of realistic traffic. In particular, the ratio of malicious and benign traffic is almost the same, and the traffic volume is low.

NSL-KDD After a statistical analysis of the KDD 99 dataset, Tavallaee et al. [2009] identified issues responsible for biases that would negatively affect the performance of anomaly detection mechanisms. The NSL-KDD dataset attempts to alleviate these issues by creating a new dataset from specifically chosen records of the KDD 99 dataset.

UNSW-NB15 Motivated by the lack of publicly available and modern datasets that reflect modern network traffic scenarios, Moustafa et al. [2015] created the UNSW-NB15 dataset in 2015. The dataset was generated using a mixture of network test beds and synthetic traffic generation tools. Nine different attack families are included, all of which are labeled. The dataset is provided in the PCAP format as well as a record of network features (in the same style as the KDD 99 dataset).

IRSC Created in 2015 by the Indian River State College (IRSC), this dataset consists of real controlled and uncontrolled attacks carried out in a production network [Zuech, Taghi M. Khoshgoftaar, et al., 2015a]. The datasets consists of full network traces in the PCAP file format along records of network flows. Labels are provided for all controlled attacks while some uncontrolled attacks are labeled through

the usage of the Snort IDS. The dataset does not look to be publicly available as of now.

NGIDS-DS Haider et al. [2017] proposed a metric to evaluate the realism of datasets for IDS. Taking into account the qualities that improve their proposed metric, they developed a synthetic and labeled dataset with a medium-high score (according to their metric). The dataset is provided in the PCAP file format along with labels and the logs of each network host. The dataset emulates five different network scenarios: e-commerce, military, academia, social media and banks.

3.3.1.1 *Requirement Compliance of Static Datasets*

In Table 3.1, we summarize the compliance of the static datasets we survey against the requirements we propose in Section 3.2.1. The columns of the table correspond to each proposed functional and non-functional requirements for static datasets (except the last column). Our last non-functional requirement, that of *quality*, is replaced with a different one as quality does not lend well to a summarization. In Section 3.3.3, we discuss in detail the quality of different datasets. The *genuineness* attribute of a dataset replaces the *quality* requirement. A genuine dataset is one that is generated using traffic of a real network environment. A synthetically created dataset is not considered *genuine*. In the datasets we survey, we found that the *genuineness* of a dataset correlates with its quality: Synthetic datasets contain more issues or defects that genuine ones. Therefore, the *genuineness* attribute works as a heuristic that determines the quality of a dataset.

No single dataset satisfies all our proposed requirements. This attests to the difficulty of creating useful datasets for this field of research. Up to now and according to our requirements, the NGIDS-DS dataset is the most suitable one as it is labeled with the ground truth, contains full package information, is flexible (incorporating multiple network scenarios), and is distributed in the PCAP file format. It lacks the *renewable* requirement and *genuine* attribute. The lack of renewability is not currently a problem as the dataset is modern (published in 2017); however, the dataset will eventually become outdated. The genuineness attribute is a more subtle topic. The community has strong negative feelings against synthetic datasets. Nonetheless, it is arguable that, for a system to perform well in real network environments, the system must also perform well using synthetic datasets [Abt et al., 2013].

3.3.2 *Dataset Generation Tools*

Static datasets are difficult to maintain and update. This is evident from the fact that most static datasets, as seen in Table 3.1, have troubles meeting the *renewable* and *flexible* requirement. More often than

Dataset	Payloads	Labels	Ground Truth	Renewable	Flexible	Public	Interoperable	Genuine
DARPA 98/99	✓	✓	✓	✗	✗	✓	✓	✗
KDD 99	✓	✓	✓	✗	✗	✓	✗	✗
MAWI	✗	✓	✗	✓	✗	✓	✓	✓[s]
CAIDA	✓	✗	✗	✓	✓	✓[a]	✓[i]	✗
LBNL	✓	✗	✗	✗	✗	✗	✗	✓
Kyoto	✗	✓	✗	✓[k]	✗	✓	✗	✓
SimpleWeb	✗	✓[l]	✗	✗	✗	✓	✓[i]	✗
UMass	✓	✓[l]	✗	✗	✓	✓	✓[i]	✓[s]
IMPACT	✗	✓[l]	✗	✓	✓	✓[a]	✓[i]	✓[s]
CDX	✓	✓	✓	✗	✗	✗	✓	✓
NSL-KDD	✓	✓	✓	✗	✗	✓	✗	✗
IRSC	✓	✓[l]	✓	✗	✗	✗	✓	✓
UNSW-NB15	✓	✓	✓	✗	✗	✓	✓	✓[t]
NGIDS-DS	✓	✓	✓	✗	✓	✓	✓	✗

[a] Access is restricted or special permissions are needed.
[i] The PCAP file format is not used all the time.
[k] Renewed up to 2015.
[l] Only partially labeled.
[s] Mixed set of genuine and synthetic data.
[t] Genuine background traffic with synthetic attacks.

Table 3.1: Summary requirements of 14 different static datasets.

not, when researchers need modern datasets targeted at specific scenarios, they build custom tools to generate them. In this section, we present two tools (similar in scope to our proposal) for creating custom datasets and an overview of our tool (ID2T). Afterwards, we summarize how the tools conform to the requirements we propose for tools that create synthetic traffic.

FLAME Brauckhoff et al. [2008] have developed a tool for injecting anomalies into network flows. The tool known as Flow-Level Anomaly Modeling Engine (FLAME) injects anomalies using one of three different modes. With an additive mode, flows are injected without modifying the background traffic (e. g., port scans). Using the subtractive mode, flows are removed from background traffic (e. g., ingress shifts). Both modes are combined into an interactive mode to add and remove flows (e. g., to simulate congestion due to a Denial of Service (DoS)). Due to FLAME being discontinued, the tool can no longer be easily found in the public domain.

ISCX-UNB Shiravi et al. [2012] point out the difficulty of evaluating, comparing and deploying anomaly-based NIDS due to the lack of suitable datasets. In their work, they propose an approach to synthetically generate network traffic from *profiles*. A profile is an abstract model that describes network traffic. They created two types of models for representing profiles, one for legitimate traffic and another for malicious behavior. Legitimate traffic is modeled by replicating the statistics of real network traffic. Malicious behavior is modeled using a custom description language. A test bed is used to generate network packets from profiles. A dataset example is accessible, however, the tool is not readily available to the public.

ID2T This is the tool we develop and present in this chapter. The first prototype of this tool was released in 2015 [Cordero, Vasilomanolakis, Milanov, et al., 2015]. An improved version of the tool, and first public release, followed in 2016 [Vasilomanolakis, Garcia Cordero, et al., 2016]. The objective of ID2T is to create attacks that replicate the properties of background traffic. Background traffic is provided by the user. Attacks are manually programmed and use the statistics collection mechanism of ID2T to replicate the network properties of any input PCAP file. Many attacks are readily available that range from DoS and network scans to malware and botnet infections.

3.3.2.1 *Requirement Compliance of Synthetic Traffic Generation Tools*

There is a distinct lack of tools that can generate synthetic attacks suitable for the evaluation of NIDSs. FLAME and *ISCX-UNB* try to achieve similar goals as ID2T but do not meet the requirements we derived as essential for such tools. FLAME is not *available* online anymore despite it being originally publicly released. This is an issue that would potentially be prevented had the software been published as Free and Open Source Software (FOSS) in an open platform. Although the *flexibility* of the tool is limited, as it does not operate at the *packet level*, research that specializes in processing flows would benefit from such a tool. ID2T is more general and generates network traffic at the packet level. Tools that work on flows can use the output of ID2T to extract flows with labeled attacks.

The creators of ISCX-UNB proposed a tool capable of generating synthetic background and malicious traffic. In contrast, ID2T only generates synthetic malicious traffic. The ISCX-UNB tool has the disadvantage, however, that it requires a test bed to model traffic. The test bed is not easy to setup, and requires specialized software and hardware. Due to the test bed, ISCX-UNB has low *usability*, does not *scale* well to large networks, and lacks an adequate *interactivity*. The tool is not *publicly available* and the quality of its output cannot be compared to ID2T.

Dataset	Packet Level	Interoperable	Minimal Interaction	Flexible Modeling	Scalable	Extendable	Usable	Public and FOSS
FLAME	✗	✗	✓	✗	✓	✓	✓	✗[f]
ISCX-UNB	✓	✓	✗[i]	✓	✗[s]	✓	✗	✗[d]
ID2T	✓	✓	✓	✓	✓	✓	✓	✓

[d] Only a sample dataset generated by the tool is provided.
[f] Originally publicly available. Cannot be easily found online anymore.
[i] The tool requires a dedicated test bed.
[s] It is not obvious if more profiles requires a larger test bed with more systems.

Table 3.2: Summary requirements of three different datasets generation tools.

3.3.3 *Classification of Dataset Defects*

We aim at creating a dataset generation tool that actively avoids the same mistakes found in related work. In this section, we present a classification of the defects found by others and ourselves in the datasets and tools previously presented in Section 3.3.1 and 3.3.2. Our classification is the outcome of the systematic analysis of related work. It serves as a guide to acknowledge potential defects that synthetic attacks may inadvertently leave behind. The classification also enables us to organize solutions to actively detect and circumvent dataset defects. This section is used in Section 3.6 as a guide to generate synthetic attacks that actively avoid leaving defects behind by considering this classification.

In the classification, we distinguish between two types of defects found in network traffic datasets. We use the term *artifact*, in accordance to related work [Abt et al., 2013], to refer to defects in a dataset introduced as a side effect of creating synthetic traffic. The term *deficiency* refers to a defect or problem in a dataset that originate from incorrectly using, capturing or recording real network traces.

artifact

deficiency

Figure 3.4 presents our proposed classification of defects for network traffic datasets. Defects can be divided into two general classes. The *Invalid Network Traffic* class covers defects related to the incorrect usage of network protocols or specifications (e. g., TCP communication with windows of size zero). The *Inconsistent Network Traffic* class encompasses defects associated with the existence of inconsistent or unexpected network traffic (e. g., overly regular packet interarrival times). Furthermore, in our analysis of related work, we observed that

invalid network traffic

inconsistent network traffic

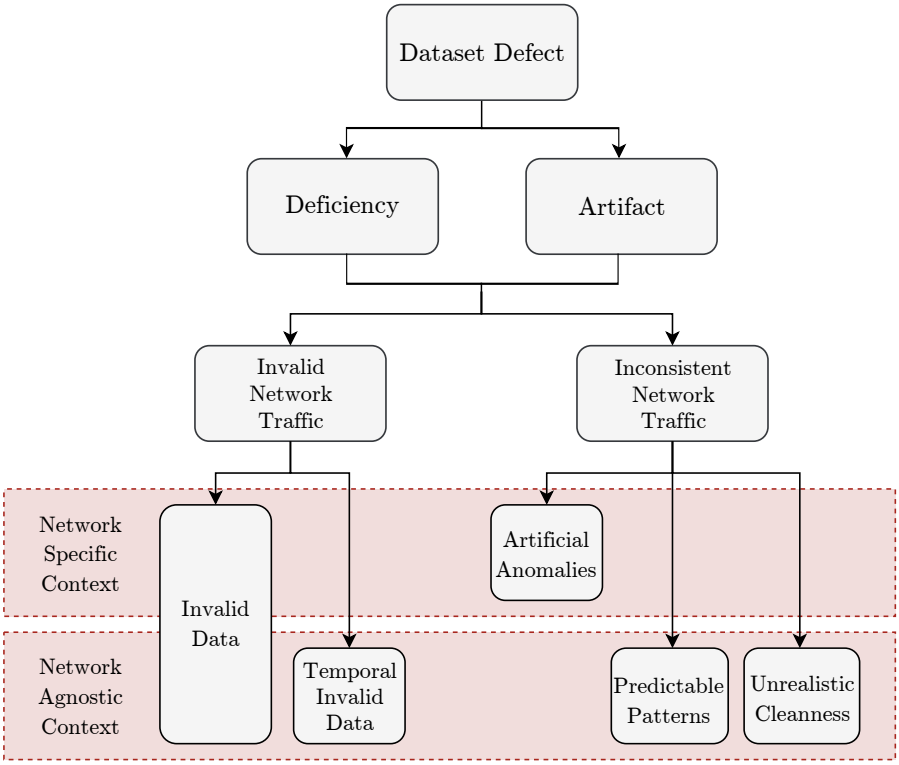


Figure 3.4: Classification of dataset defects. Defects may be distinguished as either a deficiency or an artifact. Both defect types may fall into the classes of invalid or inconsistent network traffic. These classes can further be organized into contexts depending on whether a problem is specific or agnostic to a network.

some defects may be classified as a defect only in certain contexts.[1]. Therefore, classifying something as a defect also depends on the context. From a *network specific context*, something is a defect only when the characteristics of the network traffic as a whole are taken into account (e. g., the existence of public IPs in a local network). From a *network agnostic context*, something is a defect irrespective of the characteristics of network traffic (e. g., invalid IP addresses). We use five different classes to categorize defects (see Figure 3.4). Each class is described below.

network specific context

network agnostic context

- Invalid Network Traffic
  - Network Specific Context
    - Invalid Data. Disregarded characteristics or physical limitations of a network in its captured representation.
  - Network Agnostic Context
    - Invalid Data. Violation or incorrect usage of network protocols.

1 For example, every attack packet having a TTL value of 126 is a defect only if the background packets does not have this same property.

        - Temporal Invalid Data. Incorrect usage of protocols due to protocol deprecation or renewal.

- Inconsistent Network Traffic
    - Network Specific Context
        - Artificial Anomalies. Anomalous data patterns that do not correspond to the overall characteristics of the network traffic.
    - Network Agnostic Context
        - Predictable Patterns. Network characteristics that repeat with regular periodicity, without much variance, against the expected behavior of a network.
        - Unrealistic Cleanness. Network characteristics that do not behave as expected, are too regular or do not exhibit the typical untidy behavior [Bellovin, 1992; Paxson, 1999] of network communication.

In Example 3.1, we demonstrate how our classification is used to classify some known defects of the DARPA 99 dataset. After the description of each defect, we indicate the class of the defect in parenthesis.

> **Example 3.1: Defects of the DARPA 99 Dataset**
>
> The DARPA 99 datasets contains well known defects. The TCP version used in the dataset is old [Postel et al., 1981], making the TCP header field "IPv4 Type of Service (ToS)" invalid according to modern standards [Grossman, 2002] (temporary invalid data). The dataset uses only nine different TTL values that follow an uncommon network pattern (predictable pattern). Furthermore, Matthew V Mahoney and P. K. Chan [2003] noticed that the packets of all attacks in the dataset use one of two different TTL values (artificial anomaly). As identified by [McHugh, 2000], although an attempt was made to add some of the untidy behavior typical of TCP communication, in the form of fragmented packets, the dataset does not show the expected characteristics of irregular network behavior [Paxson, 1999] (unrealistic cleanness). The data rates of the dataset are invalid if the theoretical network configuration, from where the dataset was generated, is taken into account [McHugh, 2000] (invalid data, from a network specific context).

3.4 THE INTRUSION DETECTION DATASET TOOLKIT (ID2T)

ID2T is a public and open source command-line toolkit that injects synthetic attacks into supplied (network) traffic captures. To achieve this, the toolkit first analyzes and collects statistics from a traffic capture input. The statistics are then used by attack scripts to set their

parameters and create attack packets that replicate the properties of background traffic (when appropriate). Finally, ID2T merges the attack packets with the packets of the input. The outputs of ID2T are a PCAP file with injected attacks and a file of labels that clearly indicates when attacks start and end. If needed, ID2T can watermark or taint packets to associate each packet to an attack.

Example 3.2: Using ID2T

ID2T has a command-line interface. A user executes ID2T with different arguments to control the behavior of ID2T. ID2T can be used to print different plots that relate to the statistical properties of an input PCAP:

```
$ id2t --input background.pcap --plot TTL,MSS,IPEntropy
```

ID2T can also be used to issue queries, using an SQL syntax, to obtain complex information pertaining an input PCAP:

```
$ id2t --input background.pcap --query 'SELECT ipAddress
    FROM ip_statistics WHERE pktsSent > 1000'
```

Ultimately, as its primary purpose, ID2T can inject synthetic attacks into an input PCAP:

```
$ id2t --input background.pcap --attack ddos ip.src
    =10.0.0.30 ip.dst=10.0.0.40 attackers.count=250 packets.
    per-second=400 --output output.pcap
```

3.4.1 *The Architecture of ID2T*

In this section, we present the architecture of ID2T. ID2T is a toolkit aimed at injecting synthetic attacks into PCAP files of network traffic. To blend synthetic attacks with arbitrary input PCAP files, ID2T attempts to match the statistical properties of synthetic network traffic with the statistical properties of the input. Others have validated the usefulness of replicating network properties to generate synthetic traffic [Danzig et al., 1991; Shiravi et al., 2012]. In addition to replication, ID2T enhances the quality of synthetic traffic by actively avoiding (see Section 3.6) the defects presented in our classification (see Section 3.3.3). The architecture of ID2T is therefore built to enable the two strategies of replicating statistics and avoiding defects. In addition, the architecture provides the tools needed to alter or create new attack scripts that follow both strategies.

ID2T inputs

Figure 3.5 illustrates the architecture of ID2T. ID2T processes two inputs and, using seven modules, yields three different outputs. The *inputs* include a PCAP of network traffic and a set of parameters. The input PCAP has no restrictions about its size or provenance; however,

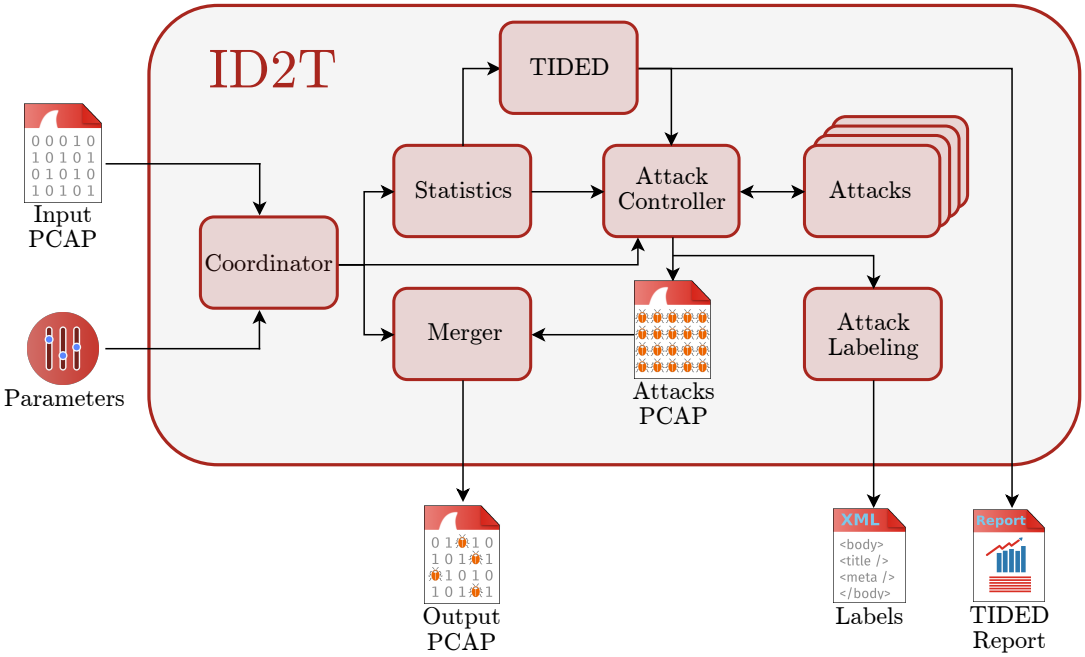


Figure 3.5: The architecture of ID2T is composed by seven modules. These modules carry out the tasks needed to create a labeled output PCAP (with its statistics) that mimics the input PCAP in accordance to some user defined parameters.

IPv6 traffic is not considered[2]. The parameters specify one or many attacks to inject and their respective properties. Example 3.3 shows how ID2T is used in the command-line to inject a port scan.

> **Example 3.3: Injecting a Port Scan Attack**
>
> The following command sets the input PCAP as *background.pcap* and a set of parameters that relate to a port scan attack.
>
> ```
> $ id2t --input background.pcap --attack portscan ip.src
>     =192.168.178.2 mac.src=32:08:24:DC:8A:72 inject.at-
>     timestamp=1536595087
> ```
>
> The IP and MAC addresses of the attacker are explicitly specified. The timestamp of when the simulated attack takes place is also specified. All other properties made available by the port scan attack script that are not explicitly specified are automatically calculated from the statistics of the input. Such other properties, for example, are the packet rate of the attack, IP address of the victim, destination ports, and status of the open and close ports, among others.

ID2T outputs

The *outputs* of ID2T include a copy of the input PCAP merged with packets created by attack scripts, a human-readable XML file with la-

2 Besides its name, IPv6 does not share many things with IPv4. The protocols function in different ways. Therefore, the research presented here does not directly translate to work with the IPv6 protocol.

bels and a report of the input statistics. The labels indicate when an attack started and finished along with all user-supplied or automatically calculated properties of the attack. Individual packets are not labeled. It is possible, however, to taint the packets (e. g., by manually specifying a known MAC address) to associate them to specific attacks. The statistics output shows a quantitative characterization of the PCAP input. This characterization includes the values of different properties such as the average packet rate, average packet size, total bandwidth used, IP address entropies, and TTL distribution, among others. When appropriate, these statistics are also reported on an interval rather than an input-wide basis. All the available statistics are detailed in Section 3.5.

ID2T modules

The *modules* of ID2T are responsible for generating the outputs from the inputs. The following section describes each module in detail.

3.4.2 *The Modules of ID2T*

ID2T is a multi-modular application built to be easily extendable. Modules are decoupled from each other and perform a specific set of tasks. In this section, we describe each of the seven modules, their tasks and how they interact with each other. Figure 3.5 illustrates each module as a small block inside the main block of the architecture.

COORDINATOR MODULE This user-facing module is responsible for parsing the command-line arguments issued by the user. The module coordinates with the *Statistics*, *Attack Controller* and *Merger* modules to perform the tasks of injecting attacks, plotting the statistics of the input or listing the properties of each attack.

STATISTICS MODULE This is a backend module responsible for collecting and computing statistics from the input PCAP file. The module collects two types of statistics, those related to the input PCAP as a whole (i. e., total number of packets), and those related to user-defined intervals (e. g., average packet rate every one minute interval). The *Controller* module enable users to directly interact with the *Statistics* module though a mode termed *query mode*. In query mode, users can query for predefined information or generate custom information queries. Example 3.4 shows how ID2T is used in query mode. The *Statistics* module also provides an interface to the attack controller and *TIDED* modules. Through this interface, the modules can query information programmatically to carry out their operations (i. e., create synthetic attacks that replicate the statistics of the input).

query mode

> **Example 3.4: Using the Query Mode of ID2T**
>
> The query mode of ID2T has predefined queries to learn more about an input PCAP file. For example, to obtain a list of the most

used IP addresses from the file *background.pcap*, the following command is used:

```
$ id2t --input background.pcap --query 'most_used(ipAddress)'
```

Users can supply their own SQL query to obtain custom information. The following command obtains, for example, all IP addresses in the input file that sent more than 1,000 packages:

```
$ id2t --input background.pcap --query 'SELECT ipAddress
    FROM ip_statistics WHERE pktsSent > 10;'
```

TIDED MODULE This is a backend module that performs qualitative tests on the input PCAP using data from the *Statistics* module. The objective of the tests is to provide measurements that may expose deficiencies (see Section 3.3.3) and sources of problems in the input PCAP file. The tests fall in one of the categories of availability, validity or diversity. A detailed description of Testing Intrusion Detection Datasets (TIDED) is given in Section 3.5.

ATTACK CONTROLLER MODULE This backend module instructs attack scripts, via the *Attacks module*, to create the packets of synthetic attacks. This module validates the attack parameters supplied by the user. It also enables the attack scripts to obtain information from the *Statistics* or *TIDED* modules. After the *Attacks module* generates packets, this module creates a consolidated PCAP file (of only attack packets) that is forwarded to the *Merger* module. Information about the attacks in the consolidated PCAP is forwarded to the *Attack Labeling* module to enable it to create a file with attack labels.

ATTACKS MODULE This module is both a user-facing and a backend module. From the side of the user-facing part, this module provides an API for users to program attack scripts. The API has functions to facilitate the programming of attacks that replicate the properties of an input PCAP file. The programmed attacks become available to the user by registering them with the *Attack Controller* module. From the side of the backend part, this module configures attacks given the set of parameters specified by the user. ID2T comes with 12 programmed attacks, each detailed in Section 3.6.

ATTACK LABELING MODULE This backend module is responsible for generating an XML file containing the labels of injected attacks. The labels include the attack name, user-specified parameters, default parameters, number of injected packets and time of injection. The file is meant to be readable by machines and humans.

MERGER MODULE This backend module merges together the attacks supplied by the *Attack Controller Module* and the input PCAP to create an output PCAP file.

3.5 TESTING INTRUSION DETECTION DATASETS (TIDED)

Quality datasets are essential to the development, comparison and evaluation of NIDSs. The overall quality of a dataset not only depends on the quality of the injected attacks but also on the quality of the background traffic. The TIDED module of ID2T performs tests on the background traffic to identify potential artifacts or deficiencies (see the classification in Section 3.3.3).

reliability tests

We assign the term *reliability tests* to the tests implemented within TIDED that attempt to expose deficiencies in the background traffic used as an input to ID2T. We borrow this term from the field of computer systems. From the perspective of computer systems, reliability refers to the probability that a system satisfactorily performs the tasks for which it is designed. Analogously, we deem NIDS databases reliable if they satisfactorily perform their intended task (provide valid grounds for the evaluation of NIDSs). Thereafter, we use the term *reliability tests* to denote tests that may uncover aspects that hamper the reliability of an NIDS dataset.

The purpose of TIDED is two-fold. On the one hand, it uses reliability tests to validate the conformance of the network traffic to some expected properties (e. g., the distribution of IP addresses conforming to supposedly backbone network traffic). On the other hand, TIDED makes the reliability tests available to the attack scripts (through the Attack Controller module) to enable a better replication of the properties of background traffic. This module confers ID2T the characteristics of a network dataset analysis tool.

We derive nine reliability tests from our survey and analysis of related work (see Section 3.3). The reliability tests are designed to detect the artifacts or deficiencies which we have classified in Section 3.3.3. The reliability tests are classified into different classes. In the following, we present the different reliability tests and their classes. Afterwards, we describe the metrics used by the reliability tests and illustrate examples of the usefulness of the tests.

3.5.1 *Classification of Reliability Tests*

We use three different classes to categorize the reliability tests of TIDED. Figure 3.6 shows a diagram of the nine implemented tests. The tests use the statistics collected by ID2T from its input PCAP file and do not need to directly analyze the file. If needed, ID2T provides an API to create new tests. The currently implemented tests are detailed next.

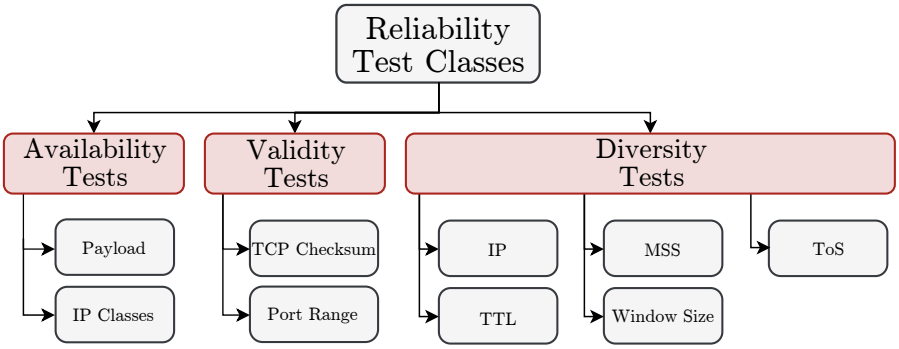


Figure 3.6: Classification of the reliability tests implemented in TIDED.

- Availability Tests. This is a class of tests that encompasses all tests that verify the existence of a property or attribute in network traffic.
  - Payload Availability. This test verifies if the packets of a network contain payloads or not. If payloads are available, the test tracks the entropy of the payloads to deduce and report on the amount of encrypted payloads.
  - IP Classes Availability. This test identifies which classes of IP addresses a network contains. The test specifies how public and private addresses mix and to what ratio.
- Validity Tests. This is a test class that corresponds to the tests that identify consistency issues within network traffic captures.
  - TCP Checksum Validity. This test identifies the amount of TCP checksum errors in the packet capture. The test reports the ratio of checksum errors.
  - Port Range Validity. This test counts the number of ports seen in the network capture for each of the three port ranges defined by the Internet Assigned Numbers Authority (IANA) in accordance to RFC 1700 [Reynolds et al., 1994] and RFC 6335 [Cotton et al., 2011]. The test reports the number of packets that target port zero.
- Diversity Tests. This is a class that relates to the tests that determine the correct or incorrect behavior of TCP header values in network communications.
  - IP Distribution. This test identifies if the distribution of source and destination IP addresses correspond to expected patterns of normality. IP address distribution correlates with the network type (e. g., home, office, or backbone network).
  - TTL Distribution. This test measures the distribution of TTL values in the TCP header field and looks for unexpected deviations from how a network behaves. This value correlates with the be-

havior of operating systems and hardware devices (i. e., different systems and devices modify the TTL differently).

- Maximum Segment Size (MSS) Distribution. This test determines if the MSS varies according to some expectation. The expectations depend on the hosts that take part in network communications. Hosts increase the MSS to maximize their output and decrease it to minimize IP fragmentation. This value correlates with the design and capabilities of a network. The MSS indirectly signals the purpose of a network (e. g., content distribution, cloud services or office work).
- Window Size Distribution. This test analyses how different the window sizes of packets are within a network capture. The window size correlates with the bandwidth of a network. Therefore, by estimating the bandwidth, the test determines if the different values of the window sizes match the expected distribution of different window size values.
- ToS Distribution. This test tracks the ToS TCP header value and detects whether the ToS follows modern protocol specifications. The meaning of these header has changed multiple times in the past, i. e., in RFC 791, 1122, 1349, 1455, 2474, 2780 and 3168 [Ramakrishnan et al., 2001]. In some specialized environments, the ToS is used to determine the priority of packets. In most environments, the ToS is ignored.

3.5.2 *Reliability Test Metrics*

The reliability tests utilize diverse metrics to characterize network traffic and identify potential sources of defects. The results of the metrics are used, both, to expose potential problems to an analyst and to better replicate the network traffic properties of synthetic attacks. We present different metrics and examples of how these are used to detect irregular network characteristics.

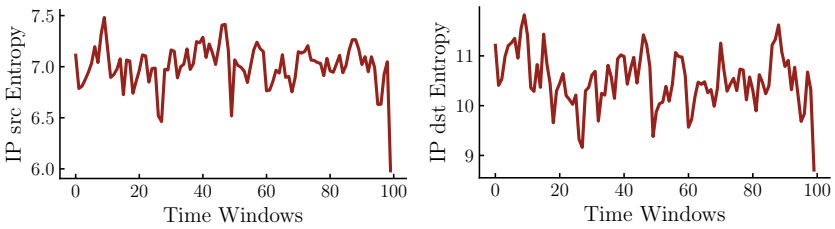
(SHANNON) ENTROPY DISTRIBUTION The entropy metric is used to characterize the uncertainty of network features within a time window. Entropy $H(X)$ is defined as

$$H(X) = -\sum_{i=1}^{n} P(x_i) \cdot \log_2 P(x_i),$$

where $X = \{x_1, x_2, \ldots, x_n\}$, x_i are values of any one feature, and $P(X)$ is the probability mass function of X. The minimum entropy is zero and the maximum is $\log_2 n$. A low entropy indicates that the values of X repeat often. Conversely, high entropy signals that the values of X do not tend to repeat. Example 3.5 shows how this metric can be used.

Example 3.5: Visualizing the Distribution of Entropies

The entropies of IP addresses provide useful information that allows us to identify noteworthy events that occur in network traffic. Figure 3.7 shows the entropies of the source and destination IP addresses of traffic in a day (2018/04/01) of the MAWI dataset. Entropies are calculated by dividing the traffic in 100 time windows and calculating the entropies of the IPs within each time window.



(a) Entropies of source IPs **(b)** Entropies of destination IPs

Figure 3.7: Measurements of IP entropies using 100 time windows on the 2018/04/01 of the MAWI dataset.

The figures make evident a noteworthy event, visible in the last time windows, where entropies drastically fall. This means that this day of the MAWI dataset cuts the traffic capturing process short. This traffic would therefore be more useful as a dataset if we disregard all packets of the last time window.

NORMALIZED (SHANNON) ENTROPY This metric corresponds to the entropy normalized in the range $[0, 1]$. The normalized entropy $H_n(X)$ is defined as

$$H_n(X) = -\sum_{i=1}^{n} \frac{P(x_i) \cdot \log_2 P(x_i)}{\log_2 n},$$

and is equivalent to dividing $H(X)$ by the its maximum value $\log_2 n$, where $X = \{x_1, x_2, \ldots, x_n\}$. This metric enables us to directly compare the values of different datasets. In Example 3.6, we compare the features of different datasets using the normalized entropy.

NOVELTY DISTRIBUTION This metric counts the different values a random variable has had at time window t_i that have not been previously seen in time windows $t \in [1, t_{i-1}]$. These counts enable us to visually determine the dynamic nature of a network. This metric correlates with the type and size of a network: In small home networks, novelty distributions are infrequently high. The contrary is true for backbone networks where the novelty distribution is not typically low.

Example 3.6: Visualizing Normalized Entropies

The properties of the static datasets presented in the related work section (Section 3.3.1) cannot be directly compared. This is because the datasets span different amounts of time (e. g., a day in MAWI spans 15 minutes while a day in DARPA spans 24 hours) and contain different packet quantities. A comparison is meaningful, however, if we use normalized entropies: All properties are characterized with entropies which we can compare when normalized. The histograms in Figure 3.8 compare seven properties of seven datasets using the normalized entropy metric.

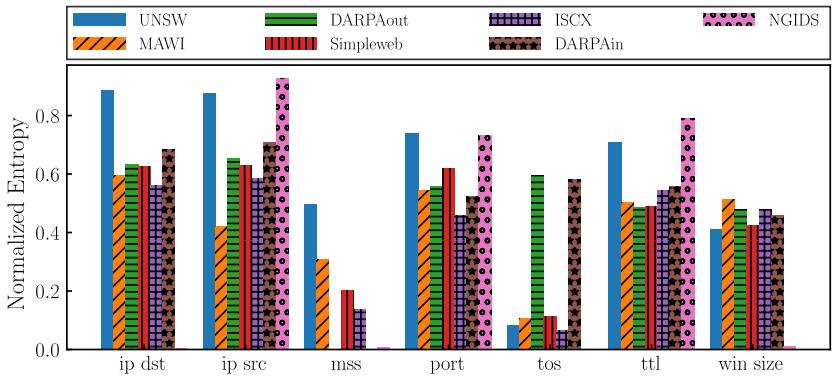


Figure 3.8: Comparing the normalized entropies of different datasets.

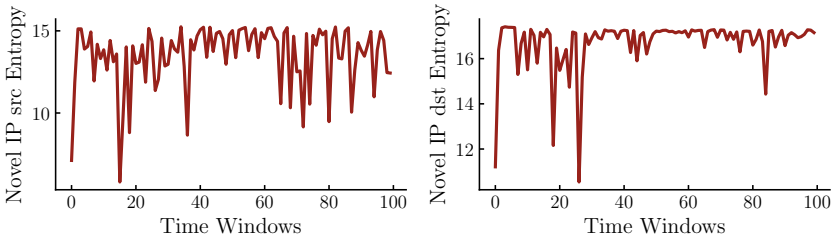
The y-axis of Figure 3.8 corresponds to the average normalized entropy of all time windows of 14 minutes of a dataset. We choose time windows of 14 minutes as all compared datasets span at least 14 minutes. *DARPAin* refers to one week of the internal traffic of the DARPA dataset. *DARPAout* refers to one week of the external traffic.

Some conclusions can be drawn from the histograms. The DARPA dataset is known to have deficiencies in how traffic uses the MSS, ToS and TTL fields [Matthew V Mahoney and P. K. Chan, 2003]. DARPA has almost zero entropy in all these fields. The fields also differ from other realistic datasets such as MAWI or Simpleweb. The NGIDS dataset shows that the normalized entropy of destination addresses is close to zero, signaling that the same IP addresses are almost always targeted. There are also no variations of the MSS, ToS and TTL fields. The normalized entropy of the ports shows that the NGIDS dataset has traffic that almost always goes to different IP addresses, something which is not typical of real users. The histograms also shows (from the high source and destination IP address normalized entropies) how the UNSW dataset captures traffic of hosts that are seen few times only.

NOVELTY DISTRIBUTION ENTROPY This metric measures the entropy of the novelty distribution metric. Entropy is typically used to estimate the uncertainty of a random variable. However, entropy can also be used to characterize the shape of a distribution. This metric characterizes the novelty distribution for easier visualizations. Example 3.7 shows some conclusions this metric allows us to make.

Example 3.7: Visualizing Novelty Distribution Entropies

Figure 3.9 shows the distribution of novelty IP addresses in a day of the MAWI dataset. From Figure 3.9b, we can appreciate that during time window 27 (spanning 9 seconds), given all newly seen IPs, only few new IPs receive the majority of the traffic. This behavior may signal traffic shifts, routing reconfigurations, the start of new alpha flows [Lassoued, 2011], network errors or other similar traffic behaviors.



(a) Entropies of novel source IPs **(b)** Entropies of novel destination IPs

Figure 3.9: Measurements of novelty distribution entropies using 100 time windows on the 2018/04/01 of the MAWI dataset.

NORMALIZED NOVELTY DISTRIBUTION ENTROPY This is a metric that normalizes the novelty distribution metric in the range of $[0, 1]$. This metric characterizes the shape of novel values and, as it is normalized, enables a direct comparison of characterizations of novelty distributions between multiple sources. An example of how this metric is used is shown in Example 3.8.

CUMULATIVE ENTROPY DISTRIBUTION This metric measures the distribution of cumulative entropies. The cumulative entropy of random variable X at time window t is the entropy of X when we take into account all the values of X until the end of t. The cumulative distribution is the histogram that results from the calculation of the cumulative entropies at every time window t. Example 3.9 shows a sample scenario where this metric gives insights into the behavior of network traffic.

Example 3.8: Visualizing the Novelty Distribution

The histograms in Figure 3.10 compare the normalized novelty distribution of seven fields of seven different datasets. Some known as a well a new insights can be deducted from the figure. Once again, we can conclude from the low novelty values that the DARPA dataset has unusual MSS, ToS and TTL values. The NGIDS dataset sees all the hosts in the dataset in few time windows, as shown from the zero normalized entropy values of its source and destination IP addresses.

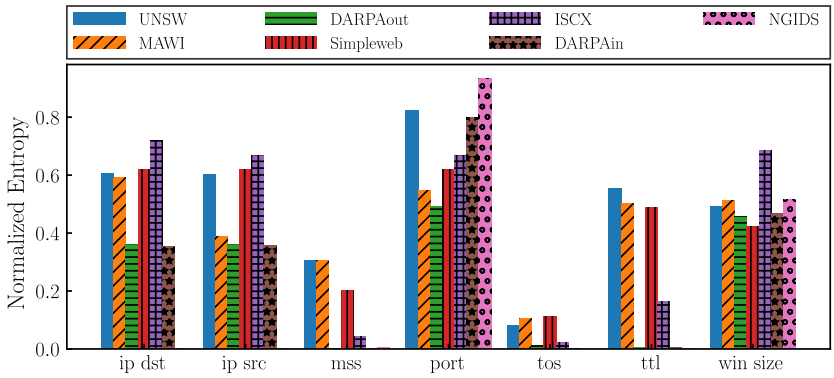


Figure 3.10: Comparing the normalized novelty distribution of different datasets.

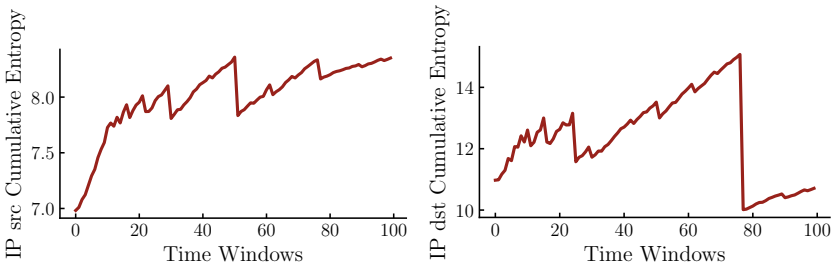
3.6 THE ATTACK SCRIPTS OF ID2T

The main goals of ID2T is to inject synthetic attacks. ID2T comes with 12 attack scripts that generate the packets to simulate diverse attacks. All attack scripts use the API of ID2T to leverage the properties of an input PCAP to disguise the synthetic nature of the packets they create. Attack scripts require different user-supplied parameters to craft network packets. If the user does not specify all parameters, ID2T selects default values for the missing ones. Default parameter values try to match the characteristics of the input. For example, if the victim's IP address parameter is not specified for an attack script that targets port 80, the script must find a suitable default. The script finds a random IP addresses within the input PCAP that receives traffic on port 80.

We classify the 12 injectable attacks provided by ID2T in four classes. Figure 3.12 shows the injectable attacks and how we categorize them. In what follows, we present four classes of attacks along with their individual attacks, the potential artifacts that may affect the classes (according to our classification in Section 3.3.3), and how we avoid artifacts.

Example 3.9: Visualizing Cumulative Entropy Distributions

Consider the cumulative entropy distribution shown in Figure 3.11. The figure highlights specific points where entropy drastically shifts; signaling swift behavioral changes in relation to past observations. In time window 78 of Figure 3.11b it is clear that the distribution of destination IP addresses drastically changed. This behavior, although not common, is expected to occur within backbone networks (which the MAWI dataset provides). If we observe this phenomenon in a home network, we would conclude that the unexpected change of traffic probably comes from a deficiency in the dataset.



(a) Cumulative entropies of src. IPs **(b)** Cumulative entropies of dst. IPs

Figure 3.11: Measurements of cumulative entropies using 100 time windows on the 2018/04/01 of the MAWI dataset.

3.6.1 *Probe and Surveillance Attack Scripts*

The attacks crafted by the attack scripts of this class, more than being explicit attacks, are probing or surveillance techniques typically used to gather information before conducting an attack. These attacks use different techniques to disclose information about the provided services or exposed vulnerabilities of network hosts. ID2T includes three different types of network probes.

3.6.1.1 *Potential Artifacts of Probes and Surveillance Attacks*

The synthetic attacks crafted by probe and surveillance attack scripts may potentially contain several artifacts. In the *invalid data* category, from a *network specific context*, these attacks could incorrectly generate an amount of packets that disregard the physical limitations (e. g., bandwidth) of the input PCAP. Determining the physical limitations of PCAP files is challenging as the PCAP file format do not explicitly carry such information. The probing and surveillance attacks that we develop estimate the bandwidth and network routing capabilities (i. e., latency and speed) to avoid creating more packets than the network can theoretically carry. Our estimates are based on the heuristic

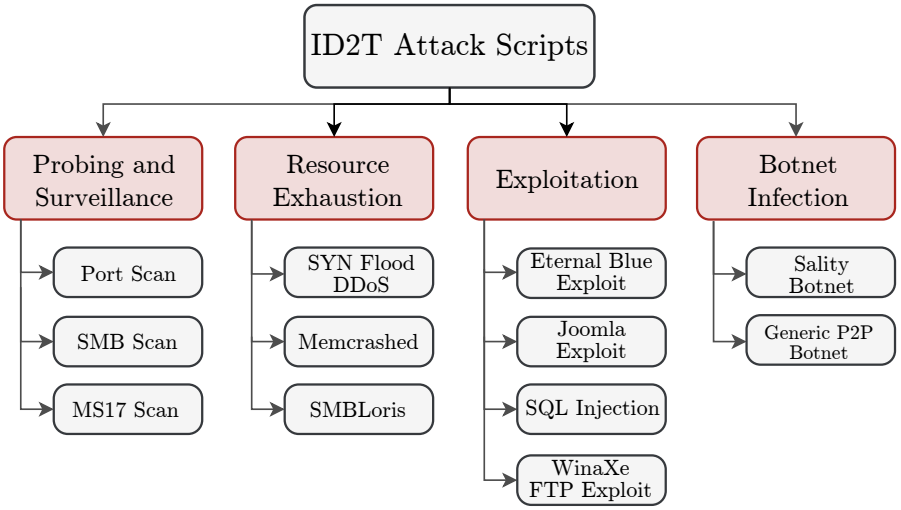


Figure 3.12: Classification of the synthetic attacks ID2T can inject. ID2T can inject 12 different attacks. We classify the attacks into four different classes.

that network bandwidth is a multiple of 10 Mb/s, and that the average packet interarrival time directly correlates with routing capabilities. Whenever our probing and surveillance attacks inject packets, they take into account the estimated physical limitations of the network and the number of packets already present: For a given time window, the attacks only inject a number of packets that does not exceed the difference between the packets present in the time window and our estimated maximum number of packets that the time window may contain.

From a *network agnostic context*, the injection of port scans and similar attacks can easily create artifacts that fall in the *predictable pattern* category. Our probing attacks consider the distribution of interarrival times and the current background traffic to estimate where and how many packets may be injected. This actively avoids injecting packets that fall within predictable time slots.

3.6.1.2 *The Synthetic Probe and Surveillance Attack Scripts*

PORT SCAN ATTACK SCRIPT This attack script emulates port scans that uncover open services in network hosts. The attack script implements a variation of this attacks that executes a vertical TCP SYN port scan: One IP address is scanned fully before moving to the next IP. The attack script disguises the synthetic nature of the packets it creates by imitating the behavior of the popular Nmap[3] port scanner. The disguise process uses an adaptive packet rate generation strategy that targets the most common 1,000 ports (by default).

3 https://nmap.org/

Example 3.10: Injecting a Port Scan Attack

The following command injects a port scan into *background.pcap* targeting IP 10.0.0.12 with a rate of 300 packets per second.

```
$ id2t --input background.pcap --attack portscan packets.per
    -second=300 ip.src=10.0.0.12
```

SMB SCAN SCRIPT This attack script generates attacks that search a network for hosts that offer SMB or Samba shared resources. The simulated attack attempts to establish a TCP connection with port 445 of a victim so as to list available shared resources. This attack script simulates the network communication that takes place when hosts respond, both, positively or negatively to SMB queries. By specifying parameters, a user can control which resources are returned as available on successful queries.

Example 3.11: Injecting an SMB Scan Attack

The following command injects an SMB scan into *background.pcap* simulating that it originates from 10.0.0.30 and targets different IPs. The IPs in the range 10.0.0.1 to 10.0.0.100 are scanned; only the IP 10.0.0.40 replies positively to the scan.

```
$ id2t --input background.pcap --attack smbscan ip.src
    =10.0.0.30 hosting.ip=10.0.0.40 ip.dst=(10.0.0.1 -
    10.0.0.100)
```

MS17 SCAN SCRIPT This is a specialized attack script that generates simulates attacks that probe hosts running the Windows operating system to determine if they have the *MS17–010* patch[4]. This patch fixes several vulnerabilities that leaves a host open to remote exploitation. The attack script disguises the attacks it generates by imitating the behavior of the open source Metasploit framework[5].

Example 3.12: Injecting an MS17 Scan Attack

The following command injects into *background.pcap* a successful scan of host 10.0.0.40 that is not patched. The scan request originates from IP 10.0.0.30

```
$ id2t --input background.pcap --attack ms17scan ip.src
    =10.0.0.30 ip.dst=10.0.0.40
```

4 https://docs.microsoft.com/en-us/security-updates/securitybulletins/2017/ms17-010

5 https://www.metasploit.com/

3.6.1.3 *Probe and Surveillance Limitations*

When a host answers back to a probing attack, the response creates many packets that may be affected by the background network traffic, attack itself or the network configuration. The network packets generated by ID2T assume that the responses follow the standard TCP procedure (using SYN and ACK control packets) without packet drops or retransmission errors.

3.6.2 *Resource Exhaustion Attack Scripts*

These attack scripts aim at conducting attacks that consume the resources of a host to deny legitimate users from using one or more services of the host. ID2T has with three resource exhaustion attack scripts.

3.6.2.1 *Potential Artifacts of Resource Exhaustion Attack Scripts*

The injection of synthetic exhaustion attacks relies on crafting large amounts of packets. This crafting process is prone to leave artifacts behind. From a *network specific context*, these attacks may introduce *invalid data* and *artificial anomalies* if care is not taken. These attacks avoid creating *invalid data* artifacts using the same technique that probes and surveillance attacks use: We design the attacks to first estimate the bandwidth and routing capabilities of an input PCAP file. Afterwards, the attack scripts create a number of packets that do not exceed the estimated physical limitations of the input (see also Section 3.6.1.1).

Synthetic resource exhaustion attacks may incorrectly inject *artificial anomalies* if the attacks craft packets that disregard the network specific characteristics of the input PCAP file. Our resource exhaustion attacks recognize and deal with anomalies that relate to IP addresses and to packet frames[6]. Regarding the IP addresses, we implement our attacks to take into account the classes (i. e., A, B C, D or E) and types (i. e., public or private) of the IP addresses present in the input. The IP of injected packets mimic these two IP characteristics. Regarding the packet frames, our attacks consider all time-related metrics of a packet (i. e., arrival time, time delta from previous packet and time of capture), packet length, payload entropy and protocol parameters (e. g., TCP flags and ports) to create packets that blend with the background traffic (when appropriate). For all considered frame properties, we fit a Gaussian probability distribution to the property and sample from the distribution to obtain values that match the background traffic.

6 The PCAP file format stores information that relates to a network packet in a data structure known as a *frame*. A frame contains packet information that goes from the Data Frame layer up to the Application layer of the OSI reference model.

From a *network agnostic context*, without adequate prevention mechanisms, the packets these attacks inject can introduce artifacts of the *predictable pattern* class. The resource exhaustion attacks of ID2T set the TCP/IP protocol properties of a packet (i. e., TTL, window size, MSS, DSCP and IP Flags) by sampling the distribution of existing properties in the background traffic. These attacks also mimic packet congestion and packet drops.

3.6.2.2 *The Synthetic Resource Exhaustion Attack Scripts*

SYN FLOOD DDOS ATTACK SCRIPT This attack script creates attacks that imitates multiple malicious network clients trying to consume the network capabilities of a host. In the generated attacks, malicious clients try to establish as many TCP connections as possible. In the attack generation process, the attack script creates the packets that would originate from attackers as well as from the expected responses of victims. The attack script uses the network activities of a victim to tune its behavior. The tuning process has two steps: First, the script determines which ports are open. Second, the script calculates a packet rate that is expected to overwhelm the incoming traffic of a victim. This attack script disguises the attacks it generates by imitating the characteristics of DDoSs generated with Metasploit.

> **Example 3.13: Injecting a SYN Flood DDoS**
>
> The following command injects a DDoS into *background.pcap*. The DDoS targets IP 10.0.0.40 and lasts 5 minutes. The victim is simulated to reply to 500 TCP connection requests before exhausting its resources.
>
> ```
> $ id2t --input background.pcap --attack ddos ip.dst
>     =10.0.0.40 attack.duration=300 victim.buffer=500
> ```

MEMCRASHED ATTACK SCRIPT This attack script creates attacks that use an amplification DoS technique that exploits a vulnerability in Memcached[7] servers to send unsolicited traffic to a victim. In the Memcached attack, an attacker sends forged UDP requests to a vulnerable Memcached server specifying the victim's IP address as the source address. The victim then receives unsolicited packets in magnitudes larger than the number of packets sent by an attacker. The attack script implementing this attack only simulates the traffic generated by the attacker. The traffic that the victim would receive is omitted. A SYN Flood DDoS attack can instead be used to simulate the traffic received by the victim.

7 Common vulnerability and exposure entry CVE-2018-1000115.

Example 3.14: Injecting a Memcrashed Amplification Attack

The following command injects the traffic an attacker would inject into a network when carrying out a memcrashed amplification attack. The file *background.pcap* is injected with packets that originate from the address 10.0.0.133 and 10.0.0.60. A precise timestamp is used to specify when the attack starts.

```
$ id2t --input background.pcap --attack memcrashed ip.src
    =10.0.0.133 ip.dst=10.0.0.60 inject.at-timestamp
    =1457823600
```

SMBLORIS ATTACK SCRIPT This attack script creates synthetic DoS attacks that exploit a vulnerability[8] in the SMB protocol to exhaust the resources of a system. With the SMBLoris exploit, an attacker forces large memory allocation on a system which renders the system inoperable.

Example 3.15: Injecting an SMBLoris DDoS

With this command, ID2T injects a SMBLoris attack into *background.pcap* that is simulated to target IP address 10.0.0.30. A total of 25 attackers are simulated to take part in the DDoS attack. ID2T chooses the IP addresses of the attackers automatically such that the generated packets blend with the background traffic.

```
$ id2t --input background.pcap --attack smbloris ip.dst
    =10.0.0.30 attackers.count=25
```

3.6.2.3 *Resource Exhaustion Limitations*

Resource exhaustion attacks change network flow patterns. The most prominent changes include, but are not only limited to, increased response times, network congestion, increased network latency and high packet-loss rates. The attack scripts of ID2T do not alter the background traffic. Instead, attack scripts add additional traffic to existing traffic. ID2T therefore cannot replicate the side-effects of resource exhaustion attacks. Most NIDSs, however, rely on detecting attacks from suspicious traffic rather than from the observable side-effects of unsuspecting traffic.

3.6.3 *Exploitation Attack Scripts*

These are attacks that target specific defects and vulnerabilities in software with the goal of executing unwanted code in a target system. ID2T provides five different attack scripts of this exploitation class.

8 A common vulnerability and exposure entry is not available.

The attack scripts focus on replicating the real payload contents and characteristics of an exploit. In this type of attacks, the contents of the payload are as important as the properties of the TCP/IP headers as most NIDSs detect these attacks through deep packet inspection.

3.6.3.1 *Potential Artifacts of Exploitation Attack Scripts*

To create synthetic exploitation attacks, we supply the exploitation attack scripts of ID2T with the packets generated by real exploitation attempts. The scripts use the packets as templates and modify them to match the network properties of the input PCAP and the user parameters. Modifying existing packets to make them look as if originating from a different network may leave diverse artifacts behind.

From a *network agnostic context*, we take care that the attack scripts do not alter packets such as to create artifacts of the *invalid data* class. When we modify real packets, we ensure that the resulting packets conform to the underlying protocols and network specification of the input. The exploitation attack scripts of ID2T, when needed, recalculate the TCP, IP and Ethernet packet checksums. For each packet of a template, the attack scripts also modify and set adequate TCP/IP flags; and modify the classes and types of IP addresses to be consistent.

The process of generating synthetic exploits from packet templates can create artifacts of the *predictable patterns* class. We take two measures to avoid these types of artifacts. First, the exploitation attack scripts of ID2T alter the order of the established connections in the template (without affecting the order of the packets) and change the interarrival times between the packets. Second, the scripts employ different examples of successful and unsuccessful exploitation attempts which are used depending on the conditions of the background traffic. These two measures ensure that different injections of the same attack (in the same or different PCAP) do not produce the same network trace.

3.6.3.2 *The Synthetic Exploitation Attack Scripts*

ETERNAL BLUE EXPLOIT ATTACK SCRIPT This attack script creates attacks that exploit a buffer overflow vulnerability in the SMB protocol implementation of the Windows operating system[9]. The imitated attack consists in crafting a especially designed SMB message which an attacker can use to take control of a host. This attack script of replicates the exploit as if launched with the popular Metasploit framework. The created attack relies in the creation of many TCP connections, all of which the script replicates and adapts according to user-supplied parameters.

9 Common vulnerability and exposure entry CVE-2017-0144.

Example 3.16: Injecting an Eternal Blue Exploit

The following command injects traffic as if an eternal blue exploit took place in the network capture contained in *background.pcap*. ID2T chooses an IP address for the attacker and victim that blends the attack with the background traffic. The packets that belong to an attacker are watermarked with the MAC address 11:22:33:44:55:66. This enables us to follow every individual attack packet in the resulting PCAP file.

```
$ id2t --input background.pcap --attack eternalblue mac.src
    =11:22:33:44:55:66
```

JOOMLA EXPLOIT ATTACK SCRIPT This attack script forges attacks that execute arbitrary code in servers hosting the Joomla[10] content management system. The attack being imitated consists in exploiting a vulnerability[11] that enables remote attackers to create user accounts with full privileges. This attack script uses packets created by the Metasploit framework as a template. The attack script manipulates the TCP/IP and HTTP headers of the template packets to replicate properties of background traffic and adjust to user-supplied parameters.

Example 3.17: Injecting a Joomla Exploit

The next command injects traffic into *background.pcap* that looks like a Joomla remote exploit. The injected attack is made to look like originating from the IP address 10.0.0.54 and is injected after 500 packets are seen in the background.

```
$ id2t --input background.pcap --attack joomla ip.src
    =10.0.0.54 inject.after-pkt=500
```

SQL INJECTION ATTACK SCRIPT This attack script forges the packets that an SQL injection attack would create against a vulnerable[12] ATutor[13] learning management system. The replicated attack creates special SQL commands that are sent over the network to a system hosting the ATutor platform. These sent commands bypass the authentication of the system and give administrator privileges to an attacker. This attack script replicates the behavior of sending malicious SQL commands over a network. To disguise the synthetic nature of the created attacks, the attack script uses the packets created by Metasploit when it is used to conduct a real SQL injection attack against ATutor. The attack script uses the parameters supplied by

10 https://www.joomla.org/
11 Common vulnerability and exposure entry CVE-2016-8870.
12 Common vulnerability and exposure entry CVE-2016-2555.
13 https://atutor.github.io/

the user and the input background traffic to blend and disguise, respectively, the synthetically generated packets with the background traffic.

Example 3.18: Injecting a Memcrashed Amplification Attack

This next command injects a synthetically created SQL injection attack into *background.pcap*. The attack targets the port 8080 of a victim. ID2T automatically chooses the IP addresses of the attacker and the victim to match the background traffic.

```
$ id2t --input background.pcap --attack sqli port.dst=8080
```

WINAXE FTP EXPLOIT ATTACK SCRIPT This attack script creates the packets of an attack that target the WinaXe FTP client. The attack script imitates an exploit that gives full unauthorized control of the client's system to a third party. The exploit uses a buffer overflow vulnerability[14] to corrupt the memory of a WinaXe FTP client and give control to whoever controls the malicious FTP server. The attack script gives users the possibility of specifying the payload that the attack contains.

Example 3.19: Injecting a WinaXe FTP Client Exploit

With the next command, ID2T injects a synthetically created attack that simulates the exploitation of a WinaXe FTP client. As a parameter, the user specifies *script.bat* as a file that the payload of the attack carries.

```
$ id2t --input background.pcap --attack ftpwinaxe custom.
    payload.file=script.bat
```

3.6.3.3 *Exploitation Limitations*

Exploitation attacks are characterized by network traffic that transports malicious code in the payload of packets. ID2T replicates this type of attacks accurately by using the packets generated by real-world tools as templates. However, ID2T can only replicate the behavior of an attack instance. For example, in an Eternal Blue exploit, the established TCP connections vary according to many variables. With ID2T, a fixed number of connections are generated.

3.6.4 *Botnet Infection Attack Scripts*

These attack scripts create synthetic traffic related to botnet infections. A botnet is a collection of compromised systems that respond to the

14 A common vulnerability and exposure entry is not available

commands of an unauthorized actor. Botnets are mostly used to conduct malicious activities that require the coordination of many systems. Typical examples of these malicious activities range from email spam and advertisement fraud to DDoS attacks. ID2T includes attack scripts that inject traces of a real or hypothetical botnet.

3.6.4.1 *Potential Artifacts of Botnet Infection Attack Scripts*

Botnet infection attacks have characteristics and patterns similar to resource exhaustion and exploitation attacks. Consequently, generating synthetic botnet infection attacks may create the same artifacts as those potentially generated by resource exhaustion and exploitation attack scripts. These attack scripts avoid artifacts of the *invalid data* (from the *network specific context*), *artificial anomalies* and *artificial patterns* classes with the same techniques used by resource exhaustion attack scripts (see Section 3.6.2.1 for more details). In short, botnet infection attack scripts borrow the mechanisms to generate large amounts of packets from the resource exhaustion attack scripts. The scripts also adopt a template-based approach to inject botnet traffic following the mechanisms of exploitation attack scripts.

Botnet communications rely on establishing many network connections, usually using UDP as the underlying protocol. To avoid generating artifacts of the *unrealistic cleanness* type, these attack scripts simulate communication problems: packet loss, duplication and retransmission. The scripts estimate the bandwidth and network load (see Section 3.6.1.1 for details) to determine when packets are retransmitted, duplicated, dropped or lost.

3.6.4.2 *The Synthetic Botnet Infection Attack Scripts*

SALITY BOTNET INFECTION ATTACK SCRIPT This attack script creates packets as if originating from the Sality malware. The real malware infects executable files in the Windows operating system with different purposes. One of its main purposes is to give third parties unauthorized control of a systems. Systems infected with Sality communicate over a P2P network to enable the coordination and execution of remote commands. This attack script creates the network packets that the Sality variant *Win32-Sality.AM* generates when it communicates with peers. The payloads of the packets contain a malicious Dynamic Link Library (DLL) that can be found in Sality infections in the wild. The network traffic patterns and infected DLL come from VirusTotal[15]. This attack script takes care of modifying the network communication traces of Sality to match that of the background input data.

15 https://www.virustotal.com/

Example 3.20: Injecting the Traces of a Sality P2P Botnet

The following command injects the network traffic communication patterns created by a Sality P2P botnet into *background.pcap*.

```
$ id2t --input background.pcap --attack salitybotnet
```

ID2T finds reasonable defaults for all parameters: The IP address of the infected host is selected from one of the existing IP addresses of the background. The IP address of other Sality peers are chosen to be public IP addresses not present in the background (Sality avoids contacting peers in the same network). The created packets are injected at a random location where they all fit.

GENERIC P2P BOTNET INFECTION ATTACK SCRIPT This attack script generates network packets as if originating from a user-defined P2P botnet. This script simulates botnets that have multiple properties and characteristics. A user specifies the botnet communication patterns in a comma-separated file. The communication patterns specify the type and timing of the messages bots send to each other. The script transforms the communication patterns into network traffic that matches the user-supplied background traffic.

Example 3.21: Injecting a Generic P2P Botnet

With the next command, ID2T injects into *background.pcap* network communication traces that match the botnet interactions in *comm.csv*. The attack script simulates that bots with private IP addresses communicate with bots with public IP addresses through Network Address Translation (NAT). Public IP addresses in *background.pcap* are used to simulate external bots. Internal bots (with private IP addresses) are randomly created but will match the background traffic.

```
$ id2t --input background.pcap --attack memberattack file.
    csv=comm.csv nat.present=true ip.reuse.external=true
```

3.6.4.3 *Botnet Infection Limitations*

The Sality botnet variant replicated by ID2T creates network traffic that targets a set of external peers. The IP addresses of these peers are fixed and do not vary between injections. However, NIDSs identify Sality based on the network traffic footprint, which ID2T replicates, instead of the IP addresses used.

3.7 EXEMPLARY EVALUATION BY USE CASES

This section presents an evaluation in the form of two use cases. The first use case uses ID2T to reproduce the evaluation results of the anomaly-based system we present in Chapter 4 but with a different, easily replicable, dataset. The second use case validates the ability of ID2T to inject attacks with well known footprints that SNIDSs can successfully detect. In the two use cases, we use replicable datasets generated with ID2T and publicly available PCAP files (as background).

3.7.1 *Reproducing Anomaly-based Evaluation Results*

In this first use case, we demonstrate how to use ID2T to reproduce the detection capabilities of the anomaly-based NIDS we propose in Chapter 4 known as RNN. Our RNN needs background traffic (without attacks) to learn models that describe the behavior of normal traffic. We use MAWI PCAPs for this purpose (see Section 3.3.1 for a description of the MAWI dataset). To test the detection accuracy of our RNNs, we train an RNN to search for anomalies in a labeled PCAP. With ID2T, we create this labeled PCAP by injecting a day of the MAWI dataset (different from the day used for training) with DDoS attacks of different intensities.

We use four different sanitized PCAPs of the MAWI dataset to train and test our RNN. The PCAPs from the three days of the 2018/04/01 to the 2018/04/03 are used to learn a model of normal traffic. The PCAP of the 2018/04/04 is injected with attacks by ID2T and used to test the accuracy of the learned normal model. All four PCAPs go through three steps of data preprocessing. First, each PCAP is split into time windows of 10 seconds long. Second, we convert all the traffic in each time window into network flows that are then sanitized. The sanitizing process consists in removing network flows with IP addresses marked as anomalous in the accompanying labels of the PCAPs of the dataset. Finally, per time window, we calculate the entropy metric (see Section 3.5.2) of source and destination IP addresses as well as ports. An RNN uses the entropy metrics of the first three PCAPs to learn a model of normality. The entropy metrics of the last PCAP (with injected attacks) are used for testing purposes.

3.7.1.1 *Detecting Injected DDoS Attacks*

We use an RNN to detect the attacks injected by ID2T. An RNN is an autoencoder that learns to compress or reduce the dimensionality of data. In learning how to compress data, an RNN need to find a transformed space, or subspace, that fits data well. We use this subspace to determine if unobserved data is anomalous or not. If unobserved data does not fit well in the subspace found by the RNN, the data is considered anomalous in contrast to the data used to learn the sub-

space. RNNs are the core topic of Chapter 4 and we discuss them in detail there.

We evaluate the ability of ID2T to generate a useful dataset for anomaly detection by using an RNN. RNNs are mechanisms capable of recognizing DDoS attacks [Cordero, Hauke, et al., 2016]. We therefore inject a PCAP file with DDoS attacks of varying intensities using ID2T and examine how well an RNN can identify the attacks. Furthermore, we show how the RNN does not find unexpected anomalies besides the injected attacks.

To test ID2T, we train an RNN using the PCAPs from 2018/04/01 to 2018/04/03 of the MAWI dataset. With the subspace learned by the RNN, we analyze the PCAP from 2018/04/04 when it is injected with DDoSs of varying intensities. We denote this injected PCAP as the *testing* PCAP. An RNN uses the entropy metrics of IP addresses and ports within a time window to calculates anomaly scores. Whenever the anomaly score of a time window is above a threshold, we consider that an attack took place in that time window. We set the threshold as the maximum anomaly score observed during training of the RNN.

testing PCAP

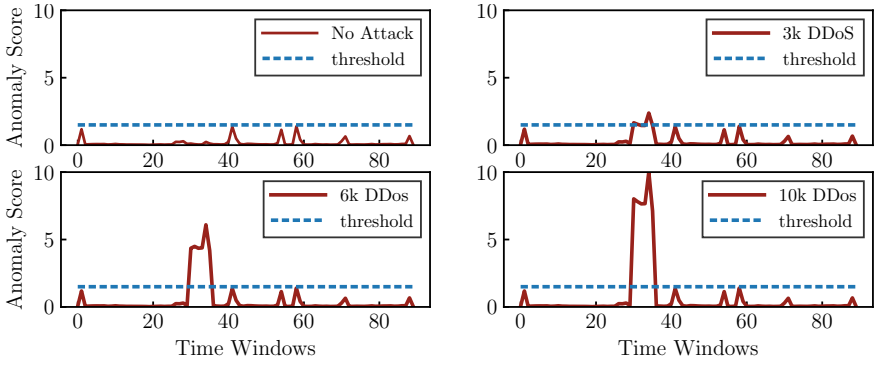


Figure 3.13: Using a RNN to detect DDoS attack injected with ID2T. The top left figure is a reference that shows the anomaly score of a PCAP that has not been injected with attacks. The other three figures show the anomaly scores of the same PCAP injected with different DDoSs of different intensities.

Figure 3.13 shows the anomaly scores of the testing PCAP when we inject it with DDoSs of different intensities. The upper left plot serves as a reference: It shows the anomaly scores of the testing PCAP at different time windows when no attacks are injected. The other three plots show DDoSs attacks that distribute 3,000; 6,000 and 10,000 packets throughout time windows 30 to 36 of the testing PCAP. Considering that the testing PCAP has 78 million packets in total, the percentage of added packets corresponds to 0.23, 0.46 and 0.77 percent, respectively. Despite the relatively small footprint of the injected attacks, the anomaly scores of all attacks lie above the threshold and are therefore detected. Outside of the attacks, no other anomalies are registered. This last fact implies that ID2T is not creating artifacts by mistake outside of the areas injected with attacks.

With ID2T, we are able to easily recreate the evaluation results of RNNs we later present in Chapter 4. Although the evaluation conducted in this section uses the same MAWI dataset as our evaluation of RNNs in Chapter 4, the dataset used in this evaluation is two years more recent. This demonstrates how ID2T can be used to reproduce and replicate the results of other NIDSs without relying on the exact same dataset.

3.7.2 *Validating Signature-based Configurations*

In this exemplary evaluation by a use case, we use ID2T to create PCAPs with injected attacks and determine if two SNIDSs detect the injections. We choose attacks that leave well-known signatures that should be identified in a correctly configured NIDS. Each attack is injected into the same background PCAP (without mixing them). The background PCAP consists of office network traffic collected during one minute. It contains 30 MiB of data and 55,000 thousand packets.

Table 3.3 shows the detection results of the Bro [Paxson, 1999] and Suricata[16] NIDSs on four different PCAPs (all having the same background traffic but different injected attacks). The injected attack is shown in the left-most column of the table. A check mark (✓) indicates that the NIDS successfully identified the injected attack; otherwise, a cross mark (✗) is shown.

	Bro	Suricata
Port Scan	✓[a]	✗
EternalBlue Exploit	✓	✓[b]
WinaXe FTP Exploit	✗	✗
Sality Botnet	✓	✓[c]

[a] Identified as: Scan::Port_Scan 188.165.214.141 scanned at least 15 unique ports of host 188.165.214.253 in 0m4s
[b] Exploit identified as: ETERNALBLUE MS17-010 Echo Response and ETPRO TROJAN (possible Metasploit payload)
[c] Sality identified as: ETPRO TROJAN Win32/Sality.AM and ET MALWARE Sality Virus User Agent Detected (KUKU)

Table 3.3: Testing two different SNIDSs against four different attacks injected by ID2T. A check mark (✓) indicates that the attack was detected; otherwise, a cross (✗) is used.

The results are not indicative of the detection capabilities of NIDSs. Instead, the results demonstrate the effectiveness of the set of signatures or anomaly rules we installed in the NIDSs. The anomaly rules of Bro identify the port scan injection while the signatures of Suricata do not. The EternalBlue exploit was identified by both NIDSs due to the

16 https://www.openinfosecfoundation.org

distinctive footprint it leaves behind. Suricata additionally identified that the payload of the EternalBlue exploit originates from Metasploit (which ID2T attempts to replicate). The WinaXe exploit was not identified by neither NIDS as we did not install signatures that would otherwise detect the exploit. Both NIDSs identified the Sality botnet injection. Suricata further identified the HTTP traffic generated by ID2T that uses special headers that emulate the communication protocol of Sality. In all attacks, ID2T did not leave traces that confused the NIDSs or caused them to trigger false alarms.

3.7.3 *Discussion of the Use Cases*

The two exemplary evaluations by use cases showcase the ability of ID2T to reproduce and replicate results, and to test the detection capabilities of NIDSs. In the first exemplary evaluation, we reproduced the capabilities of RNNs using an easily replicable PCAP. The replicable PCAP comes from the same source (the MAWI dataset) as the PCAPs used in the detailed evaluation of RNNs presented in Section 4.5. Although the PCAPs are from the same source, because they have almost two years of difference, we argue that they contain significantly different traffic characteristics. This is because in the last two years backbone network traffic has significantly changed [*The state of the internet / security report* 2018]. ID2T is able to help conduct or reproduce evaluations without having to create and publish a dataset from scratch. We use ID2T in the second exemplary evaluation to determine if NIDSs are properly configured with adequate sets of signatures or rules.

3.8 CONCLUSION AND LESSONS LEARNED

The NIDS field has had the long standing issue of not having reliable, modern and adequate datasets to develop, compare and evaluate systems. Most researchers face the problem of choosing between public outdated datasets or privately-created modern datasets. Because network data is prone to carry sensitive information, private datasets are often not made public or are heavily anonymized. Heavy anonymization implies that network payloads are removed and IP addresses are scrambled. This anonymization procedure yields datasets that lack the basic requirements we identify as essential to effectively evaluate NIDSs.

We proposed ID2T to address the issue of creating public, reliable and reproducible datasets. Instead of producing a single dataset that would eventually become outdated, ID2T allows researchers to create multiple labeled datasets that contain diverse attacks. ID2T injects synthetic attacks into user-supplied PCAP files. To avoid creating unintentional defects when injecting synthetically generated traffic, ID2T

replicates the traffic characteristics of the supplied PCAP. Additionally, it actively avoids creating the same mistakes found in other datasets.

We engineer ID2T taking into account the results of systematic scientific work. The idea to develop ID2T comes from our analysis of related work and our inability to obtain recent datasets. In our analysis, we conclude that current datasets cannot be effectively used to evaluate many NIDSs. Thereafter, we derived a set of requirements that datasets need to be useful for the evaluation of NIDSs in a general sense[17]. We design ID2T to try and meet all requirements. However, we found that the *quality* requirement of datasets was difficult to address. To make ID2T create quality datasets, we classified and grouped all defects found (by others and ourselves) in publicly available datasets. Afterwards, we used our defect classification to design the attack scripts of ID2T. The attack scripts are responsible for replicating background traffic properties as well as actively avoiding the introduction any of our classified defects. Finally, we evaluated the ability of ID2T to create quality datasets using an exemplary evaluation by use cases.

3.8.1 *Future Work*

We wish to expand the capabilities of ID2T into two directions. First, we plan to continue developing the ability of ID2T to create synthetic attacks that are indistinguishable from real attacks. Second, we look into addressing the main limitation of ID2T: In its current form, ID2T adds network traffic into a background PCAP without modifying the background traffic. We plan to create a new ID2T module that enables a feedback loop between the *attack controller* and the *merger* modules to alter the input traffic to better match the characteristics of an attack. The creation of such a feedback module will require a careful study that explores how the properties of normal traffic are affected by an attack. In essence, we will need to predict the changes that an attack would introduce into already existing traffic by only looking at the desired parameters of an attack.

ID2T is only concerned with the creation of synthetic attack traffic; the user is expected to supply their own input PCAPs to serve as background traffic. With the current advancements in the ML field of Generative Adversarial Networks (GANs), we are looking into giving ID2T the ability to create synthetic background traffic that is indistinguishable from certain types of background traffic.

17 Many datasets are only useful in specific contexts and for a limited set of NIDSs. A dataset is useful in general if it can be used by any type of NIDS.

3.8.2 *Chapter Summary*

This chapter presented ID2T, a toolkit we developed to inject synthetic attacks into user-supplied PCAPs. The objective of ID2T is to create datasets that meet the requirements needed to develop, evaluate and compare NIDSs. These requirements were laid down in Section 3.2. A survey of available datasets was then carried out taking into account the requirements defined in Section 3.3. No single dataset met the requirements needed. Furthermore, many datasets contained defects that were uncovered by researchers and our analysis. We created thereafter a classification to categorize the defects, shown in Section 3.3.3. We then proceeded to present ID2T and its modular architecture in Section 3.4. The architecture of ID2T enables a user to control how and when attacks are injected. ID2T is then responsible for blending the synthetic traffic it generates with the user's input and for labeling injected attacks. TIDED is a core module of ID2T presented in Section 3.5. The module runs several tests on an input PCAP to determine if the input has quality issues. Such tests allowed us to identify public datasets with characteristics that make them unsuitable to evaluate NIDSs. In Section 3.6, we classified and described each attack available to ID2T (along with their limitations). Finally, we presented two use case scenarios in Section 3.7. The use cases demonstrated how ID2T can be used to easily replicate datasets to evaluate NIDSs, and to evaluate the effectiveness of the rules or signatures installed in SNIDSs.

4

INTRUSION DETECTION

CONTEXT

With their size and growth rate, large networks are difficult to secure due to the amount of data they transport, among many other reasons. The challenge is further exacerbated by the presence of sophisticated, coordinated and distributed attackers. Researchers have proposed NIDSs to detect coordinated attacks, yet single NIDSs do not scale to large networks. This is mostly due to the amount of information NIDSs need to process coupled with the limitations of signature-based and supervised-based NIDSs. With the recent advancements in neural networks and unsupervised ML techniques, anomaly-based NIDSs are slowly making progress towards becoming scalable to large networks.

Developing NIDSs that work on large networks was difficult as procuring suitable datasets for testing was a major challenge. In the previous chapter, we presented ID2T and associated concepts that enables us to create labeled datasets out of arbitrary PCAP files. With ID2T, we can inject attacks into publicly available PCAPs of large networks to simulate different types of coordinated attacks. With the ability to create suitable datasets, we can now develop an NIDS that is properly tested and evaluated.

The focus of this chapter is to present an anomaly-based NIDS applicable to large networks that can detect coordinated resource exhaustion attacks (e. g., DDoSs attacks) and profiling techniques (e. g., port scans). We use ID2T to generate datasets to train RNNs, a type of autoencoder and unsupervised neural network, to detect network anomalies. With our anomaly-based NIDS, we identify attacks even when the footprint of the attacks is relatively small in relation to the background traffic.

The overview shown in Figure 4.1 puts this chapter into perspective. It shows the contribution of this chapter in relation to all others and places it in the context of the CIDS architecture we reference (see Section 2.3.3). In relation to the other chapters, the anomaly detection mechanism we propose herein employs the output datasets of Chapter 3, *Dataset Generation*, to learn models of normality that detect distributed attacks. Our mechanism is described and evaluated in this chapter in its centralized variant. We do not realize or evaluate a decentralized version in this chapter as the crucial issues of decentralization are handled elsewhere (see Chapter 5). However, the

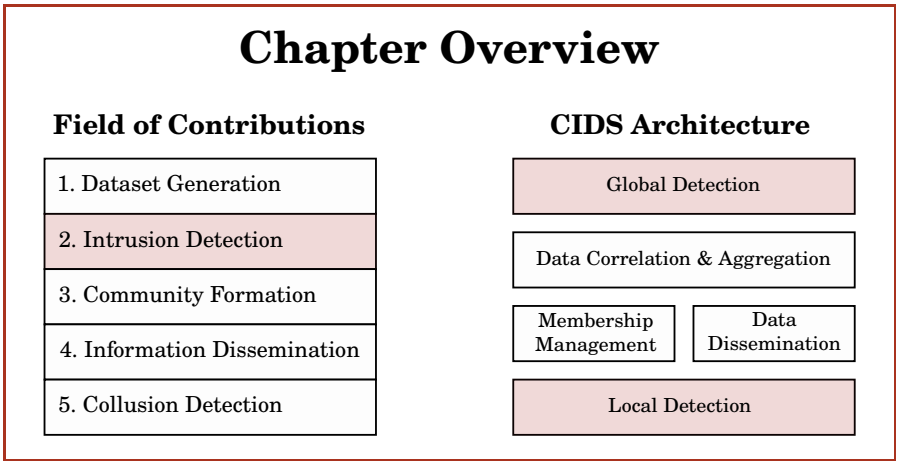


Figure 4.1: This chapter comprises the second contribution of this thesis: *Intrusion Detection*. The contribution is tied to the highlighted layers of our referenced CIDS architecture: *Local Detection* and *Global Detection*.

concepts of our centralized mechanism are directly applicable to a decentralized variant if we use the community formation mechanism laid out in Chapter 5.

In relation to the CIDS architecture we reference, this contribution chapter is related to the lowest and highest layers of the architecture, i. e., the *Local Detection* and *Global Detection* layers. At the local layer, detection relies only on local network traffic to yield intrusion alerts. At the global layer, detection relies on aggregated and correlated alerts from the local layer to yield network-wide alerts. The anomaly-based NIDS we propose has no theoretical limitations that prevents it from detecting intrusions at either layer. We, however, only explicitly address the problem of detecting intrusions at the local layer, i. e., we only consider network traffic. There is, however, no theoretical limitation that prevents our mechanism from working at the global layer.

This chapter is organized as follows. To understand our methodology to detect anomalies in network traffic, we introduce how network flows are characterized with entropy (Section 4.2.2) and the concept of RNNs. Afterwards, we elaborate on how these three topics come together to create anomaly detection models for network data (Section 4.4). Building such models implies extracting entropies from packet captures (Section 4.4.2) and training and validating RNNs with the extracted entropies (Section 4.4.3). After establishing the methodology, we present evaluations that support RNNs as viable anomaly detectors in large networks (Section 4.5). In the evaluation process, we explain the used dataset (Section 4.5.1), describe how it is sanitized and how we establish ground truth to test the accuracy of our model (Section 4.5.1.1). As part of the evaluation, we explain the details of the experimental setup (Section 4.5.2) and present our results (Section 4.5.3). Finally, we present a discussion of our experiments

(Section 4.5.4) and general conclusions that include future work (Section 4.6).

4.1 INTRODUCTION

The most widespread network communication protocols used these days were devised more than 20 years ago. At that time, securing network communications against resource exhaustion attacks or profiling techniques was not a concern. Now that large networks are constantly subjected to widespread coordinated attacks, network operators need to face the difficult challenge of detecting attacks that barely leave traces. In fields with the problem of analyzing vast quantities of data, researchers propose supervised learning mechanisms as possible solutions, e. g., deep learning. However, it is impractical to use supervised learning, where labeled data is needed, to detect attack in network traffic. Labeling network traffic is expensive as it tends to be vast and not readily amenable to human comprehension. Automatic labeling is also not possible due to the non-stationary behaviour of network traffic. Therefore, we are in need of intrusion detection methodologies that use unsupervised learning and do not require labeled data.

4.1.1 *Problem Statement*

Mechanisms that detect distributed attacks within large networks need to cope with many challenges. We need mechanisms that detect the types of attacks that leave small and disperse network footprints within millions of packets that originate from thousands of sources. Former signature and supervised-based NIDSs, with their memory and processing limitations, are often overwhelmed by the vast amounts of information they need to process to detect the signatures or anomalous patterns of distributed attacks. With the recent advancements in neural networks and unsupervised ML techniques for network traffic characterization, anomaly-based NIDSs are becoming usable within the problem space. For example, NIDSs can use neural networks to distributedly learn models to detect patterns while unsupervised ML techniques can learn compressed representations of network traffic that significantly lowers memory requirements.

Working directly with the network packets generated in large networks is prohibitively expensive. Instead of directly analyzing network packets, NIDSs that specialize in the analysis of large networks need to work with network flows [Lakhina, Crovella, and Christiphe Diot, 2004]. Network flows summarize the packets that have been transferred between two end systems in a certain period of time. We therefore need anomaly-based NIDSs that works with network flows.

4.1.2 *Challenges*

If anomaly-based NIDSs want to be usable within large networks, they need to overcome four main challenges. First, they need to scale to large amounts of network traffic. Second, they need to be accurate while minimizing the rate of false positives. This is because even low false positive rates may translate to large amounts of false alarms in the context of large networks. Third, the anomaly detection models of an NIDS should not require labeled data for training. Labeling network traffic is expensive even in small networks as network traffic alone is hardly comprehensible to humans. Finally, anomaly-based NIDSs need to cope with the fact that, when learning models of normality, they cannot expect to work with completely normal and sanitized network traffic.

4.1.3 *Chapter Contributions*

In this chapter, we propose an anomaly-based NIDS that can cope with the four challenges just described. In particular, we propose to use RNNs to detect network-wide attacks in large networks. RNNs use traffic feature distributions, from network flows, characterized by their entropy. We leverage the autoencoder and dropout concepts of deep learning to create the models of normality required by anomaly-based NIDSs (see Section 2.2.3). RNNs provide a methodology that has the potential of scaling with the number of analyzed features. RNNs are resilient to learning models where the training data has highly anomalous learning samples that should not be integrated into a normality model. Finally, RNNs can be used in real time as they are fast. They both learn and predict in only a matter of seconds with standard commodity hardware.

4.2 SPECIALIZED BACKGROUND

This section details the key topics needed to understand how we use RNNs to develop anomaly detection systems that detect distributed attacks in large networks. At its core, our mechanism uses *network flows* to learn a model of normality that later enables us to detect abnormal network flows. However, we do not directly work with network flow features. Instead, we work with features of *network flows characterized by their entropy*. Furthermore, the dimensionality of these features is reduced into a smaller *subspace*. The reduction, or transformation, is performed using RNNs. These topics are detailed in what follows.

4.2.1 *Network Flows*

Network traffic from large networks comes in vast quantities and has a complex nature. In its raw form, vast traffic quantities cannot be used to efficiently compute meaningful normality models that accurately describe traffic [Lakhina, Crovella, and Christophe Diot, 2004]. Hence, dimensionality reduction and feature extraction techniques are necessary [Lakhina, Crovella, and Christophe Diot, 2004]. Additionally, network traffic, as seen by packet capture tools, is a combination of heterogeneous data types. For example, source addresses constitute categorical data, packet sizes constitute numerical data, and TCP SYN flags may be described with the ordinal data type. To cope with the challenges of dealing with heterogeneous data, to effectively use complex data, and to build computationally feasible models of normality, researchers aggregate network data into network flows.

Network flows are representations of network data that summarize the communication process of two end-points during some period of time. Flows are defined, according to RFC 3917, as the tuple $F = (K, t_s, t_e, A)$ [Quittek et al., 2004]. The element K is called the flow key. The key is composed of five attributes, the source and destination IP, the source and destination ports, and the communication protocol used by the end-points. The time when a flow is first observed is t_s, and the time when a flow is finished or deliberately cut short is t_e. A is the set of aggregated information elements associated with a flow. The information $a \in A$ can span summary statistics, network payload data or event flags, among others. Common examples of information gathered from flows are the size of the transferred data, the set of all TCP flags observed during a communication session or the total number of network packets.

The process of collecting flows is carried out by networking devices or flow metering tools. Many modern network devices operating on layer 3 are capable of aggregating network packets into flows in real time. When PCAP files are instead available, a flow metering tool can be used to extract flows. Regardless of the method used to extract flows, suitable parameters need to be chosen to export flows. The *idle timeout* parameter specifies a time after which a flow with no traffic activity is exported. The *active timeout* parameter determines when a flow is exported even if traffic activity is still being observed. These parameters are especially important when time windows are used to build traffic feature distributions, as explained next.

idle timeout

active timeout

4.2.2 *Characterizing Network Flow Features with Entropy*

Network flows are widely used to reduce the dimensionality of network traffic. This, however, is not often sufficient when the amount of network traffic is significant. Gigabit network links observe millions

of network flows in the span of a minute, and each flow can be represented with hundreds of dimensions [X. Li et al., 2006]. By computing the entropy of traffic feature distributions during a window of time, we can reduce the complexity and dimensionality of network flows. This computation also transforms all network features into numerical values which we can then use as inputs to an RNNs (as they only work with continuous values).

Representing feature distributions with their entropy is an adequate approach to cope with the challenges of dealing with the heterogeneity and complexity of network traffic. Entropy characterizes with one single value the shape of a distribution. If a distribution is uniform, the entropy will be close to its maximum possible value. If the probability mass of a distribution collapses onto one single value, the entropy will be close to zero. Let us consider the example of a DDoS attack. If we analyze how the destination IP addresses are distributed during this attack, we observe that the observation of one or few IP addresses will overwhelm all others. As such, the entropy of the destination IP address feature will be reduced in contrast to previous observations. If we consider, on the other hand, a port scan, because the diversity of contacted ports suddenly increases, the entropy of the destination port feature increases in relation to previous instances where no port scan took place. Example 4.1 illustrates how entropies can characterize a distribution of IP addresses.

Multiple types of entropy can be utilized to characterize the distribution of network flow features. Lakhina, Crovella, and Christophe Diot [2005] uses the Shannon entropy while Tellenbach et al. [2011] the Tsallis entropy. We exclusively use the Shannon entropy in this chapter as the Tsallis entropy has the disadvantage of having a tunable parameter. In our work, we aim at creating an unsupervised method that does not require adjustable parameters.

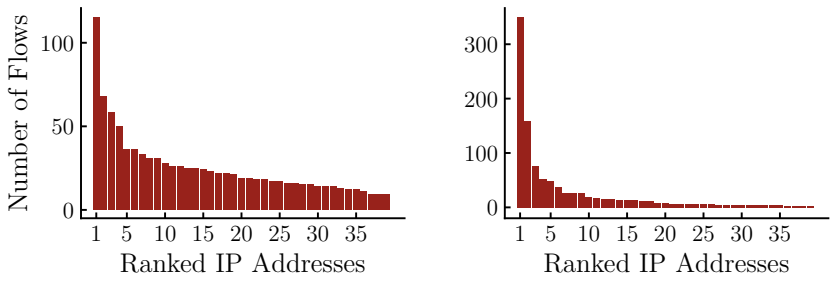
4.2.3 *Known Subspace Method for Detecting Anomalies in Network Flows*

subspace method

The *subspace method* is a type of data embedding technique [Roweis et al., 2000] that uses a dataset to learn a mathematical subspace to represents normal data and another to represent abnormal data. It uses Principal Components Analysis (PCA) [Wold et al., 1987] to compute the Principal Components (PCs) of a dataset. A normal subspace is defined as the Cartesian coordinate system composed by the PCs that capture the most variance of the dataset. Accordingly, the abnormal subspace is the Cartesian coordinate system defined by the PCs not used for the normal subspace. The number of PCs used for the normal subspace are chosen such that a desired amount of variance is accounted for. In our work, we use RNNs to compute subspaces and use PCA as a metric of comparison.

Example 4.1: Entropy Capturing Characteristics of Distributions

Figure 4.2 shows histograms of all different IP addresses observed in an office network sorted by their counts during some time window. Figure 4.2a shows the distribution of the IP addresses under normal conditions. Figure 4.2b shows the distribution when the network experiences a DoS attack. When a DoS takes place, the feature distribution of destination IP addresses is skewed towards the left to accommodate the counts of the packets targeting the attacked address. As a consequence, the entropy value during the attack period is reduced. This implies that during a DoS attack the uncertainty of observing different IP addresses is reduced as one or more victims are targeted by packets with increased regularity.



(a) Counts of observed IP addresses while traffic is normal.

(b) Counts of observed IP addresses while a DoS attack is taking place.

Figure 4.2: Exemplifying the characterization of distributions using entropy. Two distribution are shown for the counts of IP addresses observed in an office network during a normal and abnormal time windows.

4.2.4 *Replicator Neural Networks*

replicator neural networks

RNNs are neural networks, and particular instances of autoencoders [Deng et al., 2014], that were originally proposed as a data compression technique [Hecht-Nielsen, 1995]. RNNs learn to reconstruct their inputs after compressing them. By using an intermediate layer with fewer units than the input units, compression is achieved [Hecht-Nielsen, 1995]. This compression process is proven to be related to the PCA dimensionality reduction technique [Hecht-Nielsen, 1995] that subspace methods use. The standard RNN has five layers, including the input and output layers. An example RNN architecture is shown in Figure 4.3. The second and fourth layers are fully connected and use either a *sigmoid* or *tanh* activation function [Karlik et al., 2011]. These two layers do not need to have the same number of units. The third later has less units than the input layer and utilizes a step-wise activation function with N number of steps [Hawkins et al., 2002].

The number of units in the input layer and output layer must be the same.

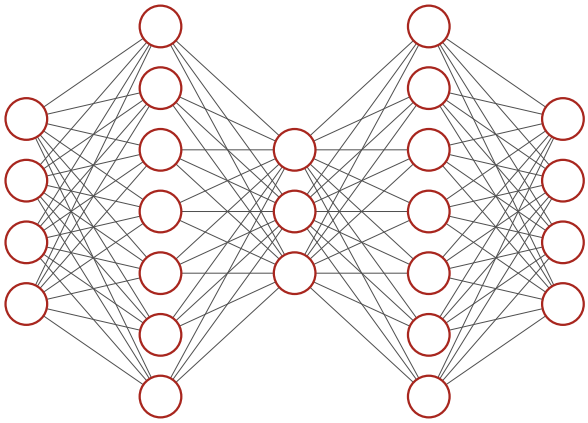


Figure 4.3: Example architecture of a Replicator Neural Network (RNN). The neural network has five layers. The first layer, that of the inputs, has four neurons. The second and fourth layers have seven neurons each. The middle layer is smaller than the input layer; it has three neurons. The last layer, that of the outputs, has by definition the same number of neurons as the input layer (four).

4.3 RELATED WORK

The related work presented in this section covers intrusion detection mechanisms based on ML that apply to large networks. We begin by describing the requirements that ML algorithms must have to qualify as applicable to this context. Afterwards, for each of the four requirements, we mention related work that addresses the requirement in relation to our contribution.

As stated in the challenges section (Section 4.1.2), anomaly-based NIDSs and, more specifically, NIDSs based on ML need to take four requirements into account to be useful in large networks. First, ML algorithms need to scale to large amounts of network traffic. Second, they need to be accurate (with low false positives). Third, they need to work without labeled data, i. e., they need to be unsupervised. Fourth, and lastly, they are required to be resilient against learning models using imperfect data, i. e., using training data that still contains attacks. The first three requirements are derived from the fact that large network traffic comes in large quantities. The fourth requirement is derived from the fact that, when using unsupervised algorithms to learn ML models, we cannot assume that large networks only process benign traffic. It is also unfeasible to pretend that traffic from large networks can be perfectly sanitized.

With regards to the scalability challenge of NIDSs, the most scalable techniques rely on learning ML models using network flows [Soysal et al., 2010]. There are many influential results demonstrating the value

of ML on top of network flows [Lakhina, Crovella, and Christophe Diot, 2005; B. Li et al., 2013; Lu et al., 2005; Nychis et al., 2008]. Network flows are common means of gathering network information as they are easy to obtain either from the perspective of a single host or multiple routers. Most, if not all routers used in large networks support some protocol specification that enables the summarization of network traffic into network flows.

Despite the challenging task of labeling network data to train the ML models of NIDSs, a lot of related work depends on supervised learning techniques. Supervised strategies using SVMs [Sallay et al., 2013] or Random Forests [Zulkernine et al., 2008], for example, need accurate labeled data to build classification models. This is a disadvantage as labeled data is hard to acquire in large networks. Our proposed methodology utilizes RNNs as unsupervised learning mechanisms that do not rely on labeled data. This contrasts with other related work, such as [G. Liu et al., 2007], where neural networks are used as supervised learning methods.

Regarding the challenge of learning normality models using imperfect data, related work uses two main approaches. The first approach consists in sanitizing data; the second relies on algorithms that tolerate learning from imperfect training data. Cretu et al. [2008] proposes a methodology for sanitizing training data for anomaly-based systems that use the data to learn models of normality. Some datasets also attempt to label potential anomalies for others to sanitize it [Cho et al., 2000; Fontugne et al., 2010]. The second approach relies on models that tolerate learning from data that contains a few mistakes or anomalies. Hartono et al. [2007] developed a supervised learning system able to learn from imperfect data. In [Breve et al., 2010], the authors develop a semi-supervised learning scheme that avoids propagating mislabeled data to generate accurate models. Our unsupervised model based on RNNs falls in the category of models that can be accurate in spite of imperfect training data.

4.4 INTRUSION DETECTION USING REPLICATOR NEURAL NETWORKS

To find anomalous network flows, we modify the RNN proposed by Hecht-Nielsen [1995] to build our own variant. With our variant, we train an RNN to replicate the entropy characterization of the distribution[1] of network flow features. Our proposed RNN uses dropout (see Section 4.4.3 for details) to create neural networks that model normal network flows without heavily relying on only few inputs. RNNs inter-

1 We use empirical distributions for this purpose. For example, in a time window t_w, we count how many times we observe each source IP address. This yields a histogram of source IP addresses which is used as the empirical distribution of source IP addresses in t_w.

nally learn normality models of network flows by first compressing the entropy characterization of network flows and then reconstructing these. This learned normality models enable us to identify anomalies and potentially malicious network flows.

In the following, we formally define an RNN. We then proceed to describe the technical aspects of how we extract features from network flows, how RNNs learn models that describe normal network flows, and how we can use the normality models to identify anomalous network flows.

4.4.1 *Formal RNN Model*

The underlying network that constitutes the RNN illustrated in Figure 4.3 can be formally defined as follows. The neural network has a total of n inputs and n outputs. In between the input and output layers, three hidden layers exist. All layers are indexed with the variable $l \in \{1, 2, \ldots, 5\}$, where $l = 1$ corresponds to the input layer and $l = 5$ to the output one. Each layer l has a total of U_l units, and we use $u_i^{(l)}$, $i \in \{1, 2, \ldots, U_l\}$ to refer to a value stored in unit i of layer l. For $l \in \{1, 5\}$, $U_l = n$; for $l \in \{2, 3, 4\}$ the number of units U_l are hyper-parameters[2] than we can freely to choose.

We use $z_i^{(l)}$ to denote the output of unit i of layer l, as explicitly shown in Equation 4.3. The output $z_i^{(l)}$ is the transformation of the value $u_i^{(l)}$ by the activation function $\phi^{(l)}(\cdot)$. The first layer of the neural network uses the identity activation function $\phi^{(1)}(x) = x$. Note that the other layers, $l \in \{2, 3, 4, 5\}$, may employ a different activation function. Some common choices for activation functions are the *Sigmoid* and the *tanh* functions, both shown as a reference in Equation 4.1 and 4.2, respectively.

$$\phi_{\text{sigmoid}}(z) = \frac{1}{1 + e^{-z}} \tag{4.1}$$

$$\phi_{\tanh}(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{4.2}$$

$$z_i^{(l)} = \phi^{(l)}\left(u_i^{(l)}\right) \tag{4.3}$$

$$u_i^{(l)} = \sum_{j=0}^{U_{l-1}} w\left(u_j^{(l-1)}, u_i^{(l)}\right) z_j^{(l-1)} \tag{4.4}$$

We calculate the value $u_i^{(l)}$ using Equation 4.4. We use the function $w\left(u_j^{(l-1)}, u_i^{(l)}\right)$ to represent a weight parameter that exists between the unit j in layer $l-1$ and the unit i of layer l. The collection of

2 The term hyper-parameter is used to refer to a variable that needs to be set in advance and that an ML algorithm considers as constant. In contrast, a parameter is a variables that an ML algorithm finds through the process of learning.

weights $W = w\left(u_j^{(i-1)}, u_k^{(i)}\right)\ \forall\ i = \{2,\dots,5\},\ j = \{1,\dots,U_{i-1}\},\ k = \{1,\dots,U_i\}$ are parameters that an RNN finds during training.

RNNs attempt to find weights W that replicate the inputs as the outputs after subjecting the inputs to expanded and compressed representations. Assume that an input of D dimensions to an RNN is given as $\vec{x} = \{x^{(1)}, x^{(2)}, \dots, x^{(D)}\}$. We define the output of the RNN, given weights W, as $f(\vec{x}_i; W) = \hat{x}_i + \vec{\epsilon}_i$. The term $\hat{x}_i$ is known as the reconstruction component of $\vec{x}_i$ and $\vec{\epsilon}_i$ is the residual, or error component, of $\vec{x}_i$. Training an RNN is the process of minimizing $\vec{\epsilon}_i$ for all possible inputs $\vec{x}_i$. This minimization process is not analytically tractable. To solve it, Stochastic Gradient Descent (SGD) techniques are used. Some examples of well known SGDs methods include Nesterov [Nesterov, 1983], Adagrad [Duchi et al., 2011] or the method simply known as *momentum* [Sutskever et al., 2013]. In our work, we use the Nesterov SGD as it proved to find the best weights that minimize the reconstruction error in our tests.

4.4.2 *Extracting Entropies*

The inputs to our RNN are fed with entropies that characterize the distribution of network flow features. The process of extracting these entropies has three steps. First, we aggregate network packets into flows in a process known as *flow export*. Second, we split flows into time windows. Third and lastly, we count the different values that network flow features take and compute their entropy.

flow export

The process of extracting entropies from network flow feature distributions starts with the aggregation of network packets into flows. Packets sharing the same flow key K, as defined in Section 4.2.1, are aggregated into bins with a length given by a *time window* parameter. We use the chosen time window value to set the *idle timeout*, the *active timeout* and the time window by which flow features $a \in A$ are aggregated and counted. All in all, this means that the process of capturing flows is synchronized with the process of counting features. As a consequence, this ensures that flow features are counted in an appropriate time window without introducing problems that arise from flows that take too long to expire, incorrectly appearing in future time windows after their actual occurrence.

time window

4.4.3 *Using RNNs to Detect Anomalies in Network Flows*

We use an RNN to create a normality model of network flows. Our RNN uses five layers despite the literature suggesting that three layers are enough [Dau et al., 2014]. Five layers allows us to use the dropout regularization technique [Nitish Srivastava et al., 2014] to yield better results. Dropout probabilistically removes units and their connections from neural networks during training. This makes the network more

resilient to co-adapting network units [Nitish Srivastava et al., 2014]. With dropout, network units avoid learning features that depend on the presence of some other feature and, instead, learn to generalize better. We apply dropout to the second and fourth layers of our RNN model.

All layers of our RNN are fully connected. The second and fourth layers use a *tanh* non-linear activation function [Karlik et al., 2011]. For the third (middle) layer, we use a standard Sigmoid activation function which has been proven to work well in this context [Tóth et al., 2004]. In formal terms, $\phi^{(l)} = \phi_{\text{tanh}}$ for $l = \{2, 4\}$; and $\phi^{(3)} = \phi_{\text{sigmoid}}$.

Our RNN is trained in rounds (known as epochs) using multiple batches. Each batch of data contains the entropies of different distributions of network flow features at different time windows. The number of features n determines the number of input neurons (and output neurons), i. e., $U^{(1)} = U^{(5)} = n$. In each epoch, a set of feature distributions characterized by their entropy are inputted into our RNN. A validation set is used to evaluate the generalization capabilities of the learning process. Once our RNN is trained, it is used as a normality model with which Anomaly Scores (ASs) are computed. ASs above a certain threshold are considered anomalous.

The set of all weights W of our RNN define the parameters of a normality model. Let us represent the set of all N training instances as $\mathcal{X} = \{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_N\}$; the entropies of D flow feature distributions (dimensions) as $\vec{x}_i = \{x_i^{(1)}, x_i^{(2)}, \ldots, x_i^{(D)}\}$; and the output of an RNN with the function $f(\vec{x}_i) = \hat{x}_i + \vec{\epsilon}_i$, as already defined in Section 4.4.1. Recall that the vector $\hat{x}_i$ is the reconstruction of $\vec{x}_i$ and the vector $\vec{\epsilon}_i = \{\epsilon_i^{(1)}, \epsilon_i^{(2)}, \ldots, \epsilon_i^{(D)}\}$ is the residuals, or error components, of the reconstruction, for $i \in \{1, \ldots, N\}$. The weight parameters of the network are updated utilizing back propagation with the Nesterov SGD method. The loss function minimized in the learning process is shown in Equation 4.5.

$$\begin{aligned} f(\vec{x}_i; W) &= \hat{x}_i + \vec{\epsilon}_i \\ L(W) &= \frac{1}{2} \sum_{i=1}^{N} (\vec{x}_i - f(\vec{x}_i, W))^2. \end{aligned} \tag{4.5}$$

As the goal of back propagation with SGD is to minimize L, the network tries to find a combination of weights W such that $\hat{x}_i \approx \vec{x}_i$ and $\epsilon_i^{(j)} \approx 0$. To avoid learning the trivial solution $f(\vec{x}_i) = \vec{x}_i$, we set the number of units of the middle layer to be less than the adjacent layers (i. e., $U^{(3)} < U^{(i)}$ for $i \in \{2, 4\}$), and use dropout [Nitish Srivastava et al., 2014] to disrupt the network by randomly shutting down units in each learning iteration (i. e., setting $u_i^{(j)} = 0$ for random values of i in some layer j).

The residual value $\vec{\epsilon}_i = \vec{x}_i - \hat{x}_i$ is used to compute an Anomaly Score (AS) that determines how anomalous $\vec{x}_i$ is. We define the AS of $\vec{x}_i$ as

$$AS(\vec{x}_i) = \frac{1}{D}\sum_{j=1}^{D}(x_i^{(j)} - \hat{x}_i^{(j)})^2 = \frac{1}{D}\sum_{j=1}^{D}(\epsilon_i^{(j)})^2. \tag{4.6}$$

To determine time windows that contain anomalous flows, we set the entropy of the distribution of features of all i time windows as $\vec{x}_i$ and calculate $AS(\vec{x}_i)$. If $AS(\vec{x}_i)$ is above a predefined threshold, we say that the time window contains anomalies. We set this threshold as the biggest reconstruction error $\vec{\epsilon}_{max} = \max(\{\vec{\epsilon}_1, \ldots, \vec{\epsilon}_N\})$ observed during training of the RNN.

4.4.4 *Detecting Anomalous Flows*

The problem of finding the flows within an anomalous time window that contribute the most to the AS of the time window is a problem with a multitude of solutions [Chen et al., 2016; Kanda et al., 2013; X. Li et al., 2006]. Therefore, we do not address the problem of finding individual anomalous flows and, instead, only concentrate on finding anomalous time windows. Although outside of our scope, we nonetheless delineate the main concepts that enable our methodology to identify the flows responsible for high ASs.

Any method capable of identifying anomalous subsets of flows can be used to single out anomalous flows using Sketches (further explained in Section 6.2.1). In their most general form, Sketches are Probabilistic Data Structures (PDSs) that store items. Sketches have r rows and each row has one associated hash and b bins. The hashes compute a value between zero and b. To store an item in a Sketch, for each row, the item is hashed with the associated hash of the row to obtain a bin number where the item is stored. Therefore, an item is duplicated r times, i. e., once per row and in only one bin of each row. For example, if we add flow F to a Sketch, the Sketch stores r copies of F in one of the b bins of each row. If we add a collection of flows $\vec{F} = \{F_1, F_2, \ldots, F_N\}$ to a Sketch, each row of the Sketch ends up with a copy of each F_i (at possibly different bins for each row). The end result is that all the bins of a row form a disjoint subsets of $\vec{F}$. In addition, the subset of $\vec{F}$ in each bin (among all rows) is different.

Let us assume that the set of flows $\vec{F}$ is known to be anomalous due to a high AS. The process of determining which flows contribute the most to the AS consists in calculating the AS of the $r \times b$ bins of the sketch. Whenever the AS of a bin (i. e., a subset of flows) is above a certain threshold, we mark that bin as containing anomalies. To find the anomalous flows, we go through every $F_i \in \vec{F}$ and determine the bins where the Sketch placed F_i. If all the bins where F_i was placed are marked as containing anomalies, we then conclude that F_i

is in part responsible for it. In practice, however, we obtain a better accuracy if we relax the restriction that all appearances of F_i must lie within an anomalous bin [Kanda et al., 2013]. This mechanism for finding flows that contribute to an anomalous AS works because a randomly sampled subset of flows retains with high probability the same traffic properties of the whole set of flows [X. Li et al., 2006].

4.5 EVALUATION

In this section, we evaluate the capabilities of our anomaly detection methodology based on RNN to detect attacks in a large network. We begin describing the evaluation dataset and proceed to test the methodology on two distributed attacks. We evaluate our capability of detecting DDoS resource exhaustion attacks and SYN port scans of different intensities. Finally, we discuss the implications of our findings and how our methodology can detect anomalies not only during testing but also while training. The evaluation also shows how our methodology can successfully learn normality models even when the training data has attacks.

4.5.1 *Evaluation Dataset*

In all our experiments, we use the MAWI dataset introduced in Section 3.3.1. We use ID2T (detailed in Chapter 3) to inject labeled attacks into the MAWI dataset and establish some ground truth. We obtain network flows from the dataset and use the destination and source IP addresses, and the destination and source ports as features. The entropy of these four features are enough to represent low dimensional manifolds that accurately describe network data [Lakhina, Crovella, and Christophe Diot, 2004]. We experimented with more features but found that indeed using the aforementioned four features yield the best results. From a different point of view, we argue that those features sufficiently describe network data as their entropies are highly correlated, as can be seen from the scatter matrix in Figure 4.4. On the non-diagonal scatter plots of the figure, the entropy of each of the four features extracted from the MAWI dataset is plotted against all other features. On the diagonal plots, we use kernel density estimations to model the individual distribution of those feature entropies. The scatter plots show that the feature entropies tend to form clusters. The diagonals show that feature entropies have a highly peaked and unimodal distribution. These two facts corroborate the argument that the four features can be represented as being situated within a low dimensionality manifold.

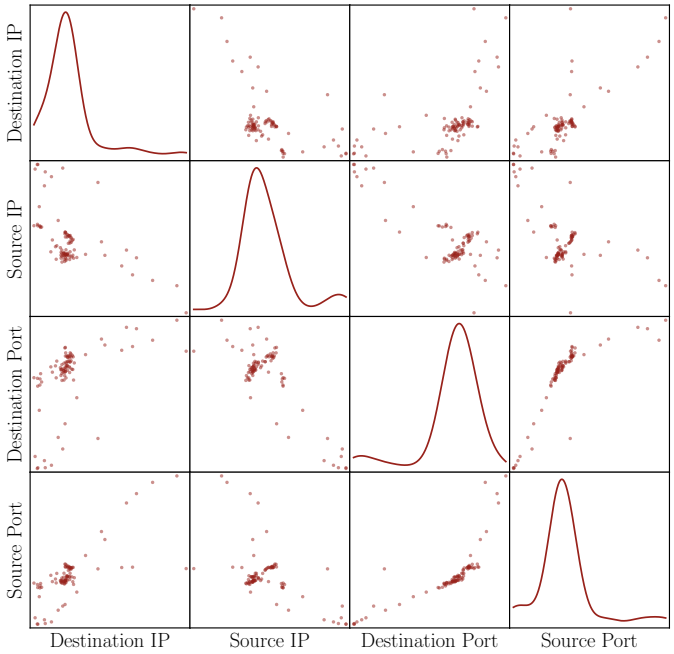


Figure 4.4: The scatter plots and distributions of flow feature entropies. We observe areas where the entropies cluster and how they correlate with each other.

4.5.1.1 *Attack Injection*

To test the accuracy of our anomaly detection mechanism, we inject DDoS resource exhaustion attacks and SYN port profiling scans into a specific time interval of a day of the MAWI dataset, or just *dataset* for short. ID2T injects attacks that mimic the network properties of the dataset and avoids creating artifacts or common defects (see Section 3.3.3).

We inject a DDoS and SYN port scan in different copies of the same dataset. We vary the intensity of the attacks by specifying the amount of packets per second to generate and the duration of the attack. In total, we test six different attack configurations consisting of three DDoS and three *SYN* port scans. All injected DDoS attacks target the IP address 90.46.2.226 and port 80. The source IP address of the attacks is chosen randomly from a set of 1,264 different IP addresses not observed in the dataset. This setup effectively simulates a DDoS botnet attack with 1,264 participating bots. All injected SYN port scans target the IP 90.46.2.226 and originate from a single host with IP 147.42.27.56. This setup simulates one machine profiling another machine.

4.5.2 *Experimental Setup*

The MAWI dataset comes in the form of PCAP files. We utilize the *yaf* flow metering tool [Inacio et al., 2010] to extract flows from the

MAWI dataset. We set the *idle timeout* and *active timeout* to 10 seconds. In all evaluations, we use the 5 days of the MAWI dataset between 2015/03/30 and 2015/04/03. The first three days are used to train models of normality. The fourth day is used for validation purposes. Different copies of the fifth day are injected with different attacks and used to evaluate the accuracy of the trained models.

We use an RNN with the architecture shown in Figure 4.3. The RNN is configured to have seven fully connected units in all its layers. The third (middle) layer uses only 3 fully connected units. These numbers were found after experimentally iterating through these parameters to find a good compromise between learning speed and convergence. We use the *tanh* activation function for the second and fourth layers, and the *sigmoid* activation function for the middle layer. Finally, we introduce a dropout of 0.5 in the second and fourth layers. This is equivalent to shutting down 50 percent of the units in the second and fourth layers in each training iteration. We tested all dropout rates in the range $[0.1, 0.2, \ldots, 0.9]$ and found that a dropout rate of 0.5 decreased the validation error the most during training.

The training of the RNN is performed using the loss function in Equation 4.5. We experimented with multiple SGD mechanisms to minimize the loss function. We consistently observed that using the adaptive Nesterov SGD [Nesterov, 1983] with a momentum of 0.2 and a starting learning rate of 0.007 (for our RNN architecture) yielded the lowest training and validation errors. We use a fixed number of epochs to train the RNN. In the following experiments, the RNNs trained during 10,000 epochs.

4.5.3 *Experimental Results*

This section presents the results of four different experiments carried out using the experimental setup just described on top of the presented MAWI dataset. In the first experiment, we demonstrate the learning capabilities of RNNs and how they can model the entropy of network flow features characterized with their entropy. In the second set of experiments, we expose the capability of our methodology to identify anomalies in the training data. Our model does not overfit and is not negatively affected by anomalies present in the training data. In the third set of experiments, we test the accuracy of our methodology by injecting attacks with ID2T and identifying their presence. Finally, in the last set of experiments, we compare our model against the *subspace method*.

4.5.3.1 *Learning Representations of Network Flows*

Our proposed RNN architecture learns accurate representations of network flows, as can be appreciated in Figure 4.5. The figure shows the average loss of the training and validation datasets during train-

ing. The validation loss is lower than the training loss as the loss of the validation function does not use regularization while the training function does. The learning process also uses three times more data, resulting in higher average loss values.

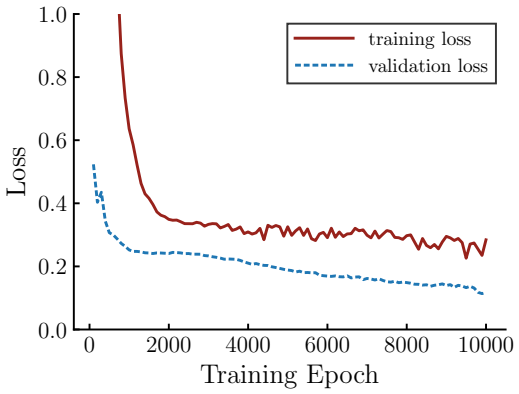


Figure 4.5: Training and validation loss during the learning process of an RNN. Throughout 10,000 iterations, or epochs, the loss of the validation dataset constantly improves. Beyond 10,000 epochs (not shown), the improvement stalls.

In Figure 4.6, we show how RNNs model normal network flows from the perspective of their AS. The x-axis covers 90 time windows of 10 seconds each. The y-axis shows the ASs of the three training and validation days. The AS is calculated using Equation 4.6. The ASs of the time windows stay within reasonable boundaries, below 0.6, except at time windows 10 and 11 of Training Day 2. As explained next, however, this is a positive effect of using dropout.

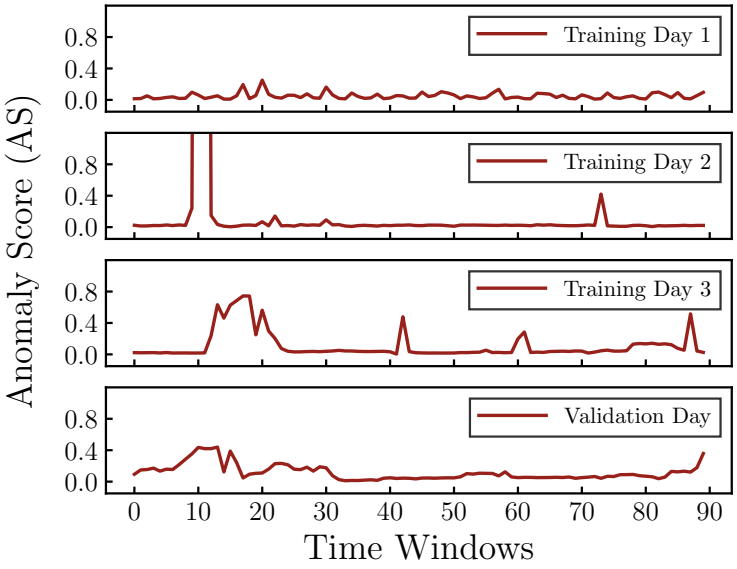


Figure 4.6: Anomaly Scores (ASs) of the training and validation days.

The traffic observed in time windows 10 and 11 of the second training day have the characteristics of an ingress shift, potentially caused by a resource exhaustion attack. In most of the training's network traffic, the number of destination IP addresses seen, during a time window, averages 200,000. In time windows 10 and 11, the number of destination IP addresses are reduced to 26,534, yet the bandwidth utilization remains almost the same. This finding also demonstrates that RNNs are able to detect anomalies during training. We further expand upon this finding next.

4.5.3.2 *Detecting Anomalies in the Training Data*

When dropout is used as described in the experimental setup (Section 4.5.2), our RNN model does not incorporate highly irregular time windows that do not follow the trend of all other time windows. Two of the time windows in Training Day 2, as appreciated in Figure 4.6, should clearly not be forcefully fit into the model. Figure 4.7 shows the ASs of the three training and the validation days. The median ASs are shown with horizontal lines and the boxes go from the first to the third quartiles. The whiskers of the plots go from the 3rd up to the 97th percentile. On the left side of the figure, we see the box plots of all the ASs of time windows of Training Day 2. There are two time windows that dominate the landscape with ASs close to 16. They correspond to the times between 2015/03/31 14:01:41 and 2015/03/31 14:02:01 of the MAWI dataset. On the right side of the figure, we present the same box plots after the anomalies are removed. The ASs of all time windows are below 0.6. Later in Section 4.5.3.4, we show that this is a quality that other methods based on PCA are not able to replicate.

4.5.3.3 *Detecting Attacks*

Our anomaly detection methodology successfully discovers all conspicuous DDoS attacks and port scans we inject. All synthetic attacks are injected on 2015/04/03 of the MAWI dataset without overlapping them. More specifically, each attack is injected into a different copy of the same day. Attacks are injected in the same spot of 14:10:33 (GMT+9) over a span of 5 seconds.

For the injection of attacks, we choose a day close to the days used for training the model. Because network traffic changes over time, the farther away the day used to test the attacks is in relation to the days used for learning, the higher the average ASs get. This means that this methodology requires a constant update of the model with recent data to better reconstruct the entropies of the network data with RNNs.

Figure 4.8 shows the ASs of the day 2015/04/03 of the MAWI dataset when it is injected with DDoSs attacks of different intensities. The top left plot shows the AS of the day without attack as a ref-

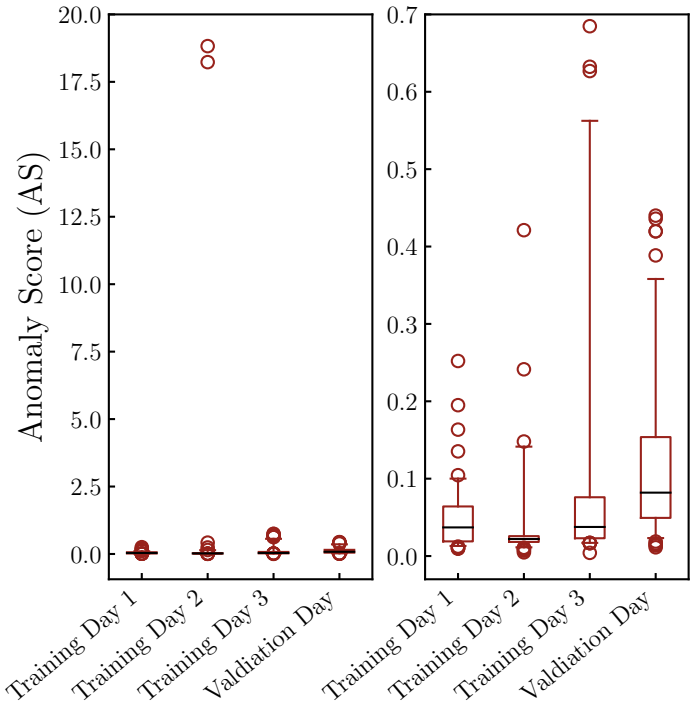


Figure 4.7: Box plots for the Anomaly Scores (ASs) of the training and validation days. On the left, two anomalous ASs are spotted. On the right, these two anomalies are removed and the plot is zoomed.

erence. The other three plots show DDoSs attacks with intensities of 1,000, 10,000 and 20,000 packets per second. With our methodology, we detect DDoSs attacks of more than 10,000 packets a second by identifying high ASs. The DDoS attack with an intensity of 1,000 packets per second is not detected. With this intensity, the DDoS attack is arguably not taxing on network devices and could be disregarded. As a justification to this statement, we refer to the benchmarks of the Linux kernel. Version 4.4 of the Linux kernel is able to process 3,500,000 SYN packets per second in commodity hardware[3].

We test three different scenarios that incorporate port scans. The top left plot of Figure 4.9 shows the AS of the day 2015/04/03 of the MAWI dataset that we choose for training when no port scans are injected. In the top right quadrant of the figure, we show the computed AS of the training day when scanning the first 1024 ports of a host in a five second interval. The bottom left and right plots show the ASs when scanning ports during a five second interval of a single host in the ranges $[1-49,151]$ and $[1-65,535]$, respectively. The two plots show that it is possible to detect port scans that scan 49,151 ports or more from looking at the AS of the time windows. Detecting port

3 The changes introduced in the linux kernel that are able to achieve this packet processing performance are found in `http://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/commit/?id=c3fc7ac9a0b978ee8538058743d21feef25f7b33`

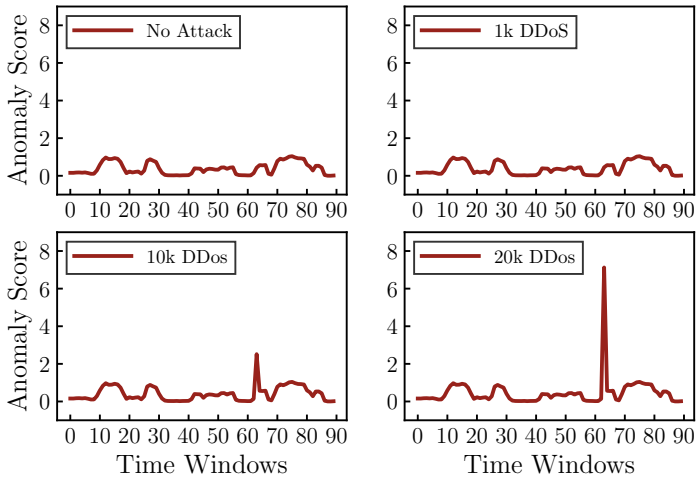


Figure 4.8: ASs calculated by a trained RNN of different DDoS attacks that have different intensities. The top left plot shows the AS of the testing day without attacks. All others plots show the AS of the testing day when attacks are injected at the same time window.

scans that scan up to the port 1024 is particularly difficult as these ports are common in the MAWI dataset. A host that scans these ports in a five seconds interval is not capable of shifting the distribution of the destination ports such that its entropy is affected.

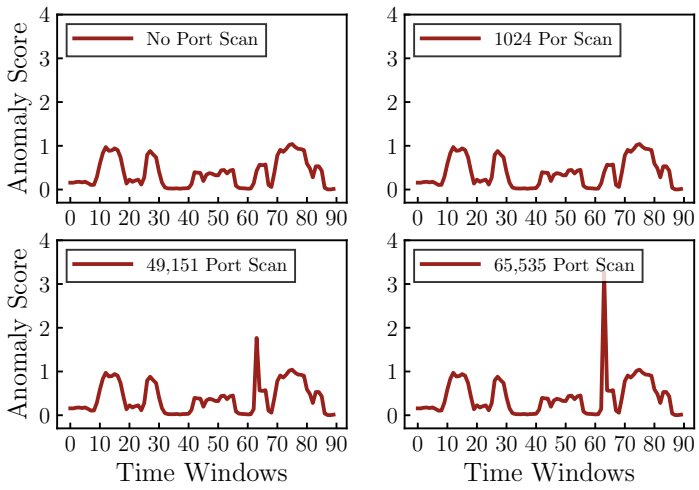


Figure 4.9: ASs calculated by a trained RNN of different port scans that go through different number of ports. The top left plot shows the AS of the testing day when there are no port scans present. All others plots show the AS of the testing day when port scans are injected at the same time window.

4.5.3.4 *Comparison Against the Subspace Method*

We use the standard subspace method (see Section 4.2.3) as a baseline of comparison. To compute normal subspaces with PCA, we use the

same four features used in the last experiments, i. e., both source and destination of IPs and ports. The days of the MAWI dataset used for training the PCA model, from which features are extracted, are also the same, i. e., from 2015/03/30 to 2015/04/01. The PCs found by PCA are shown in Table 4.1. Each eigenvalue associated to a PC relates to the proportion of variance captured. The captured variance of each PC is shown in the second row of the table. The total accumulated variance is shown in the third row. We choose PC_1 and PC_2 to represent the normal subspace as together they cover more than 90 percent of all variance in the data. PC_3 and PC_4 are used to represent the abnormal subspace. This selection of PCs yields the best results in our tests.

	PC_1	PC_2	PC_3	PC_4
Eigenvalue	1.0563	0.2062	0.0511	0.0213
Variance	79.11%	15.44%	03.83%	01.60%
Accumulated Variance	79.11%	94.56%	98.39%	100%

Table 4.1: Eigenvalues of the flow features and variance they capture.

Using the subspace method for anomaly detection is a two step process. As the first step, we project the entropies of network flow feature distributions onto the abnormal subspace. Second, we compute the norm of the projection and compare it to a threshold to determine if its anomalous or not. To determine the threshold, a statistical approach such as Q-statistic [Lakhina, Crovella, and Christophe Diot, 2005] may be used.

The subspace method detects the injected attacks that our method detects. The subspace method, however, fails to detect anomalous traffic within the training data and incorporates them into the model. Figure 4.10 illustrates this problem. The figure shows the computed norm of the entropy of network flow feature distributions projected onto the abnormal subspace for the training and validation days. The second day of training has an attack at the 10^{th} and 11^{th} time windows. However, the attack is assimilated into the model and cannot be identified with this representation (cf. Figure 4.6). RNNs, in contrast, are able to detect the attacks and avoid integrating them into the model.

4.5.4 *Discussion of the Experiments*

The datasets used in the experiments, i. e., days of the MAWI dataset, are of substantial size. The PCAPs used for training have an average of 8.6GiB in size, each with traffic rates of at least 150,000 packets per second. This is taking into account that only protocol headers

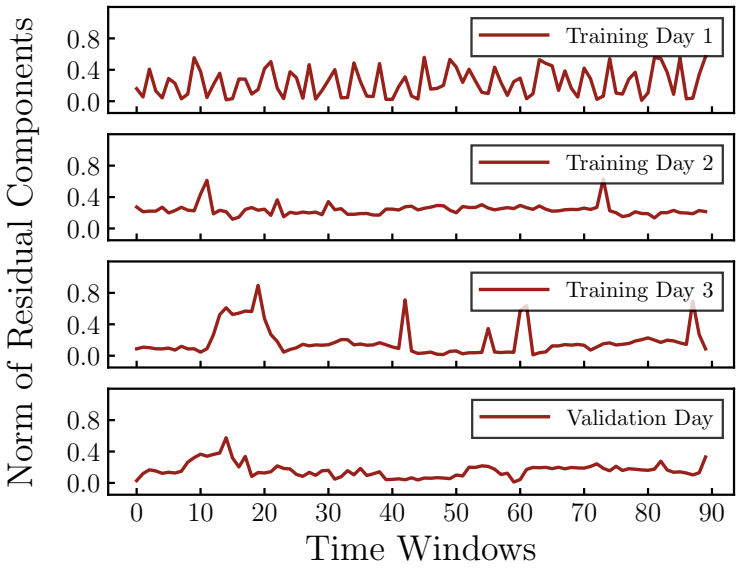


Figure 4.10: Projection of the training and validation days into the PCA's abnormal subspace. The second day of training is known to contain an attack in time windows 10 and 11. PCA undesirably incorporates this attack into the model.

are provided and network payloads are removed and that the days only span 15 minutes of time. The presented evaluation results are encouraging as the attacks injected are small in relation to the dataset used.

Our RNN based anomaly detection methodology has many advantages to other approaches. It can quickly learn and predict using large amounts of data. Learning the parameters of the RNNs takes no more than a few seconds on commodity hardware. The methodology is resilient to learning from training data that has obvious anomalies. In the learning process, the anomalies are not incorporated into the model. Removing outliers is therefore not as critical as in other mechanisms.

The presented RNNs have some limitations which can be overcome. Arguably, the most expensive part of our methodology is the extraction of features from network flows in the form of entropies. With the current flow export technology, this process can be easily automated to support single large devices or multiple devices. This, in turn, enables our methodology to be applied within distributed environments. Our methodology, as presented, requires a central component to learn the model of the RNNs. This limitation can be overcome by performing feature extraction in a distributed fashion, gathering the most significant statistics of the feature extraction process, and using a methodology that distributedly learns the parameters of neural networks from significant statistics [Lazarevic, Nisheeth Srivastava, et al., 2009]. Our mechanism detects time windows where attacks are

present; it does not exactly identify the flows that contribute to the attacks. As described in Section 4.4.4, however, our methodology can be easily adapted to identify malicious flows using Sketches (explained in Section 6.2.1).

4.6 CONCLUSION AND LESSONS LEARNED

We proposed in this chapter a methodology that uses RNNs to detect anomalies and potential attacks in large networks. Our methodology relies on two key ideas. First, we summarize the traffic of large networks with entropies that characterize the distribution of network flow features. Second, we map entropies onto low dimensionality spaces using a type of autoencoder known as RNN. Although RNNs are simple autoencoders that were once created for the purpose of compressing data, they are capable of building models of normality useful for detecting anomalies. RNNs have similar properties as PCA, another anomaly detection methodology commonly used to identify anomalous traffic in large networks. In particular, RNNs can compute normal and abnormal subspaces to model the entropies of network flow features.

Neural networks such as RNNs only work with continuous features. Network data, however, is highly heterogeneous with categorical (e. g., IP addresses), nominal (e. g., TCP flags) and ordinal (e. g., TCP SYN flags) feature types. The process of directly applying RNNs was not straightforward; we needed to transform network data into sets of continuous features. We explored many common ML techniques to transform features into continuous ones, such as *one-hot encoding*, but quickly discovered that the amount of network data from large networks made the feature transformations prohibitively expensive. From among the options presented in Section 2.1.2, we found that autoencoders that embed the entropies of feature distributions cope with the large quantities of network traffic the best.

We found that the RNN architectures used in related work [Hawkins et al., 2002] for data compression suffer from several problems when they are used to build anomaly detectors for large networks. In [Dau et al., 2014], the authors showed that RNNs of five layers can be reduced to three layers and still retain the same accuracy. We observed that dropout could not be added with positive effects in RNN architectures of three layers. Dropout was effective, however, when using a five-layer architecture. RNNs typically use a step-wise activation function in the middle layer to theoretically compress the inputs into clusters. Using step-wise activation functions is in general a problem for neural networks. The method by which neural networks update their parameters, i. e., back propagation (or gradient descent), cannot analytically compute gradients of step-wise functions to direct the search process. This is because the gradient of these functions is almost al-

ways zero. For this reason, we changed the activation function of the middle layer to the Sigmoid function, which others also found that works well in other contexts [Tóth et al., 2004].

4.6.1 *Future Work*

We see the potential of improving our methodology by using additional modern deep learning techniques given that RNN are instances of neural networks. For example, RNNs can be stacked [Deng et al., 2014], and adding Gaussian noise to the inputs of neural networks has proven to be beneficial in many situations [Deng et al., 2014]. Furthermore, RNNs may be adapted to automatically compute intermediate feature representations using techniques borrowed from convolutional neural networks [Dumoulin et al., 2016]. This would enable us to find new compositions and transformations of features that may increase the representation capabilities of RNNs.

4.6.2 *Chapter Summary*

This chapter began by introducing the problems and challenges of developing anomaly-based NIDS for large networks. To contribute a solution to the problem, we propose an anomaly detection methodology based on RNNs that can cope with the challenges and requirements imposed by large networks. To understand our contributions and methodology, we then presented specialized background that touched upon four main topics: network flows, their characterization with entropy, the subspace method for detecting anomalies in flows, and RNNs. After presenting related work related to the usage of ML for anomaly detection, the category on which RNNs fall, we proceeded to explain our methodology.

Our methodology contains two main steps. The first step consists in extracting features from network flows. Feature distributions, in the form of entropies, are extracted from network flows subdivided in time windows. At each time window, the entropies of all observed features are computed. The second step consists of projecting the extracted features (or entropies) onto a lower dimensionality space using RNNs. RNNs learn to replicate the inputs as the output after the input is forced to be compressed.

To test the capabilities of the aforementioned steps, we injected known attacks into samples of backbone network traffic and proceeded to detect these attacks. Beyond the detection capabilities of RNNs, we further explored the ability of the methodology of preventing anomalies in the training data from being included into the model. The experiments showed that RNNs can be used as efficient unsupervised mechanisms to detect anomalies in new, never before seen data, as well as the training data.

5

COMMUNITY FORMATION

CONTEXT

Distributed intrusion detection mechanisms rely on the distributed application of one or more algorithms. Algorithms typically lend themselves well to distribution if they themselves are distributed. A centralized algorithm, however, can still be used in distributed environments, such as CIDSs, if one or more central components collect data, apply the algorithm and distribute the results for merging. Collecting data is expensive due to the overhead of information dissemination, yet many algorithms benefit from using as much data as possible. This is especially true for ML-based algorithms which perform better the more data they consider. Therefore, communication overhead and detection accuracy are inversely correlated in this context.

Our intrusion detection mechanism proposed in the last chapter is of a centralized nature. With the help of neural networks, we created a mechanism capable of processing vast amounts of data to detect traffic anomalies. However, the learning process of our mechanism needs all data in one location. To implement our mechanism, or any other centralized one, within a distributed environment, we can collect data subsets in multiple locations and apply the mechanism in each location. If the algorithm is based on ML, like ours, the application of the algorithm yields a model for each location. These models can then be distributed among all locations to combine them. Models are significantly smaller in size than raw data and their dissemination is significantly less expensive than disseminating raw data.

Distributed CIDSs can establish different locations to collect data and apply centralized algorithms. Determining how many locations a CIDS would need to maximize its detection capabilities is not obvious. We need to find how to balance communication overhead and a desired detection accuracy.

A subset of CIDS sensors can use a common location to store data so that a centralized algorithm can be applied on the data. When sensors are grouped into multiple subsets for the purpose of applying a centralized algorithm in each subset, we say that the sensors form communities. In this chapter, we develop concepts that allow us to establish communities of sensors and understand how communities impact the performance of a CIDS. The communication overhead and detection accuracy of a CIDS using communities depends on the size of the communities and the number of overlapping sensors. To

test the concept of communities, we develop a distributed anomaly-based CIDS that joins the individual models built by communities. All models are joined using an ensemble learning technique.

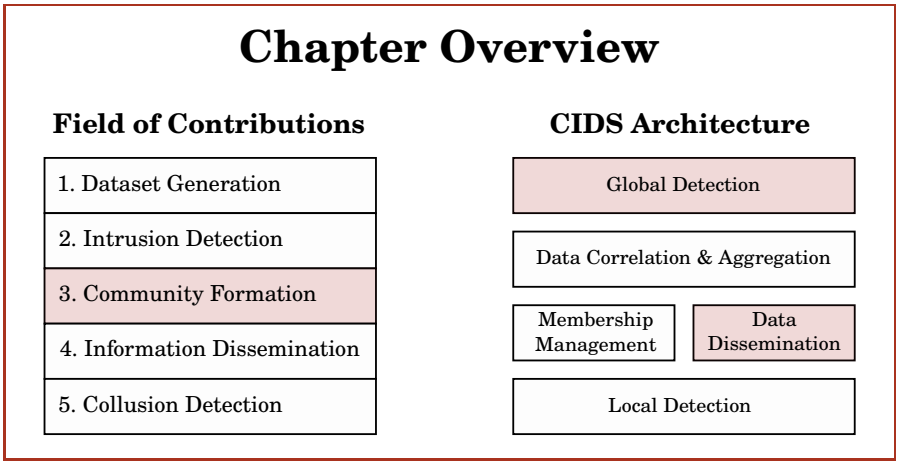


Figure 5.1: This chapter corresponds to the third contribution of this thesis: *Community Formation*. The contribution relates to the highlighted layers of our referenced CIDS architecture: *Data Dissemination* and *Global Detection*.

An overview of this chapter is shown in Figure 5.1. This chapter expands upon the third contribution of this thesis. In this contribution, we develop concepts that directly relate to two layers of the CIDS architecture we reference (see Section 2.3.3). From the perspective of the *Data Dissemination* layer, our concepts establish communication overlays that form sensor communities which balance communication overhead and detection accuracy. A CIDS uses the ML models created by multiple sensors to make decisions. This enables the system to create a detection mechanism that belongs to the *Global Detection* layer.

This chapter is organized as follows. The next section introduces the problem of using centralized intrusion detection components in distributed systems. We follow the introduction in Section 5.2 with a specialized background that summarizes the key topics needed to understand the concepts proposed herein. In Section 5.3, we present work that relates to our contribution in either the direction of anomaly network intrusion detection or distributed CIDSs. We present the concept of sensor communities and their properties in Section 5.4. The performance evaluations of a CIDS that uses our community concepts are shown in Section 5.5. The chapter finalizes with conclusion, future work and summary in Section 5.6.

5.1 INTRODUCTION

To apply centralized intrusion detection mechanisms within a distributed CIDS, the CIDS needs to assemble one or more datasets that

a centralized mechanism can process. In the context of ML, such a mechanism uses a dataset to output a model. To detect network-wide intrusions, a CIDS can merge or combine all generated models into what is known as a *meta-model*. The CIDS then distributes the meta-model among all its sensors which can then apply it within their local context to detect network-wide intrusions.

meta-model

A distributed CIDS should not rely on collecting the information of its sensor in one single location. Doing so creates a SPoF, reduces the scalability of the system and increasing the communication overhead of the network. To address these problems, a distributed CIDS can instead designate multiple sensors as locations where others can send their information. These designated sensors would then be responsible for creating datasets from what they receive, run a centralized mechanism to output a model and distribute the model to others. In the extreme case of choosing each sensor as a location where all others send their information, all information is available to all sensors without there being a SPoF. Each sensor would also be capable of creating a model that considers all information, does not need to be distributed and could theoretically achieve the best performance. However, the communication overhead would be too high and scalability would be severely constrained.

On the other side of the spectrum, each sensor could be restricted to use a centralized mechanism to compute a model using only the local data they observe, and distribute the computed model with all other sensors. Sharing models is inexpensive in comparison to sharing raw data[1]. Therefore, communication overhead is minimized and scalability is achieved. The problem of this approach is that, even after forming a meta-model from all models, we can expect that the capability of the sensors to detect network-wide intrusions is limited. The limitation stems from the fact that each individual model considers only local information.

In between the first approach of sharing information with every sensor and the second approach of not sharing information, we may consider choosing few sensors as locations where some other sensors can send their information. In such a scenario, communication overhead is influenced by not only the amount of locations where information can be sent, but also by the number of sensors that can send information to those locations. The number of locations corresponds to the number of models created by different instances of a centralized mechanism that are combined to form a meta-model. The number of sensors that contribute to the creation of a dataset (at each location) correlate with the detection capability of the model that spawns from the dataset. With the two variables of *number of locations* and *number*

1 ML models are usually no bigger than a few MiB in size while raw network data can easily span GiB in a matter of minutes. ML models are sporadically built while raw network data is constantly produced.

of sensors sending information (for each location), we can balance the communication overhead of a CIDS and the detection accuracy of a resulting meta-model.

community

We use the term *community* to refer to a subset of sensors that send information to one common sensor within the subset to create a dataset. The sensor with the dataset is then responsible for building a model using a centralized mechanism. A *community-based* CIDS is a system that integrates many communities and combines the models created by all communities to create a meta-mode. This meta model is then distributed to all communities so that they can perform intrusion detection. In this chapter, we explore the benefits, develop the concepts and study the benefits of using communities.

community-based CIDS

5.1.1 *Problem Statement*

Communities are able to influence the communication overhead and detection accuracy of a CIDS though their size and sensor overlap. Their size, i. e., number of sensors in the community, establishes the quantity of shared information as well as the completeness of the resulting dataset. The more sensors in a community, the more holistic the view of the community is and the more accurate the model it can generate (for network-wide intrusion detection). On the other hand, bigger communities leave larger communication footprints and degrade the scalability of a system. Consequently, the size of a community is one way to limit the upper bounds of the detection accuracy of the centralized intrusion detection mechanism operating within the community.

The sensors of a CIDS may belong to more than one community. We refer to these sensors as *overlapping sensors*. Overlapping sensors have the effect of duplicating their information in multiple datasets. For example, let us consider the case of combining all the datasets collected by each community. When there is no sensor overlap, the dataset combination has no repeated information. When sensors are overlapped, however, the information of those overlapping sensors is repeated in the dataset combination as many times as their overlap. When we use each individual dataset to train a model and the combination of all datasets has copies of the same data, we effectively implement the boosting ensemble technique (see Section 5.2.2.1). Boosting is the technique of creating many small datasets by sampling from a large dataset (with replacement) and creating a model with each smaller dataset. All models are combined into a meta-model and the expected accuracy of the meta-model is expected to outperform a single model created from the large dataset. This effectively means that the sensor overlap controls the degree by which boosting splits and samples a large dataset (to improve detection accuracy), but also controls some

overlapping sensors

degree of communication overhead (from sending duplicated information).

The problem at hand is to determine the effect of the *number of communities*, *community size* and *sensor overlap* in relation to the detection accuracy and communication overhead of a community-based CIDS. To test the performance of a community-based CIDS, we implement one using the well established LERAD (see Section 5.2.1) algorithm on top of the classic DARPA 99 dataset (detailed in Section 3.3.1). In spite of the dataset and the algorithm being outdated, we still use them as the algorithm is known to perform well on top of the dataset [Matthew V. Mahoney et al., 2002]. This fact enables us to test how the community parameters, i. e., their size, number and sensor overlap, influence the detection accuracy of model ensembles.

5.1.2 *Challenges*

Distributed CIDSs scale better than centralized ones at the cost of their accuracy. This can be attributed to facts such as that distributed systems cannot afford to train ML models with global information. Communities generalize the concepts of building a fully distributed ensemble model, a partially distributed ensemble model and a single centralized model (see Section 5.2.2) that can be used to detect network-wide attacks: Let us consider s sensors and c communities, where each community creates a model that participates in an ensemble of learners. The configuration of one community, i. e., $c = 1$, containing all s sensors is analogous to collecting all information in one place and using one ML algorithm to create one model. This is equivalent to a centralized system where no ensemble learning is applied. Conversely, when there are as many communities as sensors, i. e., $c = s$, and sensors do not overlap, the configuration imitates a fully distributed system where nothing is shared. In such a fully distributed system, c models are built using the local information of each of the s sensors. All c models are then combined to form an ensemble. To balance the amount of distribution, we have partially distributed ensemble models. These configurations ensemble multiple models that use information originating from multiple (potentially overlapping) sensors. Throughout this chapter, we assume that sharing models, to build ensembles of learners, has negligible network overhead costs.

alarm level

Most CIDSs work at the alarm level, e. g., [Cai et al., 2005; Gamer, 2012]. At this level, the sensors of a CIDS exchange intrusion alarms with the expectation that alarms can be correlated to detect similar attacks. Instead, we build in this chapter communities of sensors that collaborate at the *detection level*. At the detection level, data features are exchanged to collaboratively build intrusion detection models. The challenge of sending information at the detection level is that,

detection level

at this level, sending information is more costly as significantly more information is generated than at the alarm level.

5.1.3 *Chapter Contributions*

Our focus in this chapter is to develop the concepts for building communities of sensors that can leverage communication overhead and detection accuracy within distributed CIDSs. Communities of sensors enable distributed CIDSs to use centralized mechanisms without creating a SPoF. Each community is responsible for independently building their own ML model for intrusion detection. A community-based CIDS combines all learned models into an ensemble of learners to detect events that individual sensors would theoretically struggle to detect.

We explore the effects of forming communities using a well studied anomaly detector, i. e., LERAD, on top of a well known dataset, i. e., DARPA 99. Using these, we compare the detection capabilities of centralized mechanisms that learn models using global information against an ensemble of learners that use models learned from partial information. To form communities of sensors, we propose two algorithms that establish communities following constrains set by the number of communities, number of sensors in each community and maximum sensor overlap.

We evaluate the communities concept and show that an ensemble of learners that uses models created by all communities performs substantially better than isolated learners. Furthermore, we show that these ensembles approximate the best possible performance, i. e., the performance of a centralized learner, while reducing communication overhead.

5.2 SPECIALIZED BACKGROUND

This background section focuses on two key topics of crucial importance to this chapter. First, we introduce the anomaly detection algorithm we employ known as LERAD. Afterwards, we briefly describe what ensemble learning is and the boosting ensemble technique.

5.2.1 *The LERAD Algorithm*

Learning Rules for Anomaly Detection (LERAD) is a rule induction algorithm and anomaly detector that finds conditional rules that describe normal patterns in a dataset [Matthew V Mahoney and P. Chan, 2003]. The conditional rules that LERAD finds can be used to model the normal behavior of a network. A small exemplary ruleset of such a normality model is shown in Example 5.1. The anomaly detection part of LERAD uses its rulesets to identify network traffic that does not conform to rules. However, LERAD's anomaly detection mechanism is

not only based on finding network traffic that break rules, but also on identifying rare events. Matthew V Mahoney and P. Chan [2003] define a rare event as one that has not occurred after some predefined time that also has low average anomaly rates during training. Along with a decision on whether something is anomalous or not, LERAD's output also includes a confidence score.

Example 5.1: A Ruleset that Describes Normal Network Traffic

Table 5.1 shows a ruleset that may model normal network traffic. Rule 1 states that if the TCP flag of the second to last package in a conversation has the *ACK* bit set, the last flag of the packet are expected to have set either the *ACK* or *SYN* bits. Rule 2 states that if the first octet of the destination IP is "10", the second octet should be "112". Rule 3 has no condition and states that a package should only have the *SYN*, *ACK*, *PSH* or *RST* bits set. The last rule states that if a packet targets port 80 and the first word of the payload is "http", then the first word of the payloads should be "get".

Rules have an associated support and codomain. The support indicates how many data points were observed to follow the rule during training. The codomain refers to the total number of possible values that the consequent of the rule has, and it is updated during testing.

1	IF $flag_{n-1}$ = ack THEN $flag_1$ = syn OR ack	Support: 28,882 Codomain: 2
2	IF $dst.ip_1$ = 10 THEN second $dst.ip_2$ = 112	Support: 14,236 Codomain: 1
3	IF THEN $flag_n$ = syn OR ack OR psh OR rst	Support: 35,455 Codomain: 4
4	IF port = 80 AND word3 = http THEN word1 = get	Support: 12,440 Codomain: 1

Table 5.1: Example rules found by LERAD on a typical network.

5.2.2 *Ensemble Learning*

Rather that being a learning mechanism in itself, ensemble learning is a meta-model that combines multiple learned models to make predictions. Predictions are made by asking each model for its output and combining the outputs using one of many techniques. Popular

techniques include averaging all model outputs and using the most common output. In our work, for example, our combination strategy is based on using the prediction of the model with the highest confidence. The models of an ensemble may be created by different algorithms, parameters or datasets. In this chapter, we use a homogeneous ensemble where all models are created with the same algorithms. The dataset used by each model, however, is different. The most widely used ensemble algorithm that adaptively changes a dataset to create better models is known as *boosting*.

5.2.2.1 *Boosting*

Boosting is a simple and one of the most commonly used ensemble techniques to improve the accuracy of a predictor [Maclin et al., 2011]. Boosting samples a dataset with replacement to create many training sets. To improve the performance of a predictor, one predictor is trained for each training set and all predictors are combined to form an ensemble [Folino et al., 2016].

5.3 RELATED WORK

In general terms, this chapter is concerned with the development of a distributed anomaly-based CIDS that uses communities of sensors to detect network attacks. Anomaly-based systems are explained in Section 2.1.4. In this section, we elaborate on anomaly detection algorithms that build models of normality through rule induction.

5.3.1 *Rule-based Anomaly Intrusion Detection*

Anomaly detection algorithms can be coarsely grouped into three groups depending on the type of features (see Section 2.1.2) they process: algorithms that work on continuous, categorical, or mixed feature types. Network traffic features are heavily based on categorical features [Chandola et al., 2009]. Because of this reason, network-based anomaly detection is dominated by algorithms that cope with categorical features. Rule induction algorithms are a class of algorithms that exclusively work with this type.

Mahoney V. et al. [2001] developed one of the first algorithms, known as Packet Header Anomaly Detector (PHAD), that focuses on finding rules that describe the normal behavior of a network from the categorical features of network-related protocols. PHAD learns the patterns that describe a protocol and can identify network traffic that does not adhere to the learned protocols. This algorithm evolved into ALAD [Matthew V. Mahoney et al., 2002] by taking into account the application layer of the OSI model. ALAD extracts features from TCP streams to model some of the contents of network payloads. The final

iteration of this algorithm is known as LERAD [Matthew V Mahoney and P. Chan, 2003]. LERAD finds conditional rules that involve network headers and payloads to model normal network traffic.

5.3.2 *Distributed Machine Learning*

We propose concepts to form communities of sensors in CIDSs to apply ML algorithms that are intrinsically centralized in a distributed environment. Related work acknowledges two different contexts that influence how algorithms can operate on large datasets [Peteiro-Barral et al., 2013]. In the first context, large data can be moved around. In the second context, the assumption is that data cannot be moved. Our community building contribution falls on the first context. However, we further adapt our mechanism to work within a CIDS. Specifically, we focus on proposing a system that takes communication overhead into account.

Different centralized mechanisms exist to work with distributed data when data can be moved around. The importance of applying such mechanisms has gained traction and is now known as *big data* processing [Lavalle et al., 2011]. The research field of CIDSs has naturally caught on with the trend [Zuech, Taghi M Khoshgoftaar, et al., 2015b]. In [Huang et al., 2014], for example, researchers use the typically centralized latent Dirichlet allocation mechanism in a distributed environment to find network intrusions. To extract rules from distributed network data, Rodriguez et al. [2011] adapt the typically centralized genetic algorithms heuristic to work in a distributed environment.

5.4 COMMUNITY FORMATION FOR COLLABORATIVE INTRUSION DETECTION

This section presents an approach to collaboratively detect anomalies in networks using communities of sensors. We begin with a textual description of our community forming mechanisms and follow with its mathematical formalization. In the formalization process, we introduce parameters responsible for creating communities. For each parameter, we explain their impact on the community creation process and communication overhead. Finally, we propose an algorithm to choose how sensors are grouped into communities such that a chosen set of parameters are respected.

5.4.1 *The Community Formation Concept*

We wish to group CIDS sensors into communities such that a centralized anomaly-based intrusion detection mechanism can be utilized within each community. The goal is to then group the models created

by the communities to form an ensemble that can identify anomalies. Each sensor collects network flows (see Section 4.2.1) of the traffic it observes and performs feature extraction on its own. All extracted features are then sent to a central location common to all members of the community. At this central location, each community assembles a unique dataset of features that can be used as the training set of an anomaly detector, i. e., LERAD. Each different training set is used to create a model of normality, i. e., rulesets, and all models of normality are ensembled into one to make decisions (see below). The feature distributions of the training sets are expected to differ. These differences are exploited to build better predictors following the theory behind the boosting technique (see Section 5.2.2.1).

community head

Within a community, one sensor is designated to act as a central location where all other members send their extracted features. This sensor is known as a *community head*. The community head is also responsible for building a normality model using all collected features. The resulting model is then shared with all other existing community heads to form ensembles. In such a scenario, no central component is needed. We assume that sharing models has negligible impact on communication overhead as their size is diminutive in contrast to the entirety of the network traffic. We further assume that models are distributed using an appropriate dissemination mechanism (such as the one we propose in Chapter 6).

5.4.2 *Mathematical Formalization*

n_s *parameter*

n_c *parameter*

n_o *parameter*

We model our community formation concept using the graph $G = (S, E)$. The nodes S represent the sensors that belong to a CIDS and the edges E denote the connections that exist between sensors. A community is the subset of sensors $C \subseteq S$. The number of sensors in community C is denoted by $n_s = \|C\|$. The set of all communities is denoted with $\mathbb{C} = \{C_1, C_2, \ldots, C_{n_c}\}$. The total number of communities is therefore $n_c = \|\mathbb{C}\|$. Each community C_i has a sensor $s_i^\star \in C_i$ that functions as a community head, i. e., it is responsible for receiving features and learning a model. We assume that every sensor $s \in C_i$ can communicate with community head $s_i^\star$ such that $(s, s_i^\star) \in E$, $\forall s \in C_i$. As community heads need to share the models they create between each other, we assume that all community heads are connected, i. e., $(s_i^\star, s_j^\star) \in E$ where $s_i^\star \neq s_j^\star$. Sensors are allowed to belong to multiple communities simultaneously. The maximum sensor overlap is denoted with n_o, i. e., a sensor can be part of up to n_o communities. Table 5.2 provides an overview of the notation for reference. Example 5.2 gives a set of exemplary parameter settings and shows communities that conform to the parameters.

$G = (S, E)$	Community graph of sensors S with overlay E
$\mathbb{C} = \{C_1, C_2, \ldots, C_{n_c}\}$	Set of communities
S	Set of the sensors of a CIDS
$C_i \subseteq S$	The i-th community of the set $\mathbb{C}$
$s_i^\star$	Community head of community C_i
n_s	Number of sensors in a community, i. e., $n_s = \|C\|$
n_c	Total number of communities, i. e., $n_c = \|\mathbb{C}\|$
n_o	Maximum number of communities a sensor can belong to

Table 5.2: Summary of the notation used within this chapter

5.4.3 *The Community Building Parameters*

In the process of adapting a centralized algorithm into a distributed CIDS with communities, we acknowledge three parameters that influence detection accuracy and communication overhead. In this section, we discuss the influence of the number of the communities n_c, the number of sensors per community n_s, and the maximum overlap of sensors among communities n_o. Each parameter is analyzed in terms of how it affects accuracy, communication overhead and the overall scalability of the system.

5.4.3.1 *Number of Sensors in Communities*

ACCURACY The number of sensors n_s of a community greatly influences the datasets built at each community head and, therefore, has a strong impact on the detection accuracy. When $n_s = \|S\|$ (and $n_o = 1$), a centralized system is replicated as only one community exists that collect all features in its sole existing community head. The opposite of a centralized system is a fully distributed system where no collaboration takes place. This is the scenario where $n_s = 1$ (and $n_o = 1$). In such a scenario, sensors form their own community, become a community head and work in isolation. The size of n_s is bounded by $1 \leqslant n_s \leqslant \|S\|$.

OVERHEAD The overhead of n_s can be expressed as the number of edges in $\|E\|$ of graph $G = (S, E)$ between sensors s and community heads $s^\star$. The overhead can be expressed as the connections between sensors $s \in S$ and their respective community head $s^\star$. This overhead is inversely proportional to n_s. The overhead can be calculated as $\|S\| - \frac{\|S\|}{n_s}$. In this calculation, we ignore the edges that exist between community heads. Community heads only need to commu-

Example 5.2: Community Configurations

The diagrams in Figure 5.2 show how a graph G and communities $\mathbb{C}$ would be formed given different sets of parameters n_c, n_s and n_o. Configuration 1, for example, illustrates a set of communities $\mathbb{C}$ where $n_c = 2, n_s = 4$ and $n_o = 1$. Only Configuration 2 has communities with overlapping sensors, i. e., $n_o = 2$. Configuration 3 shows how the more communities there are, the more edges in E are needed.

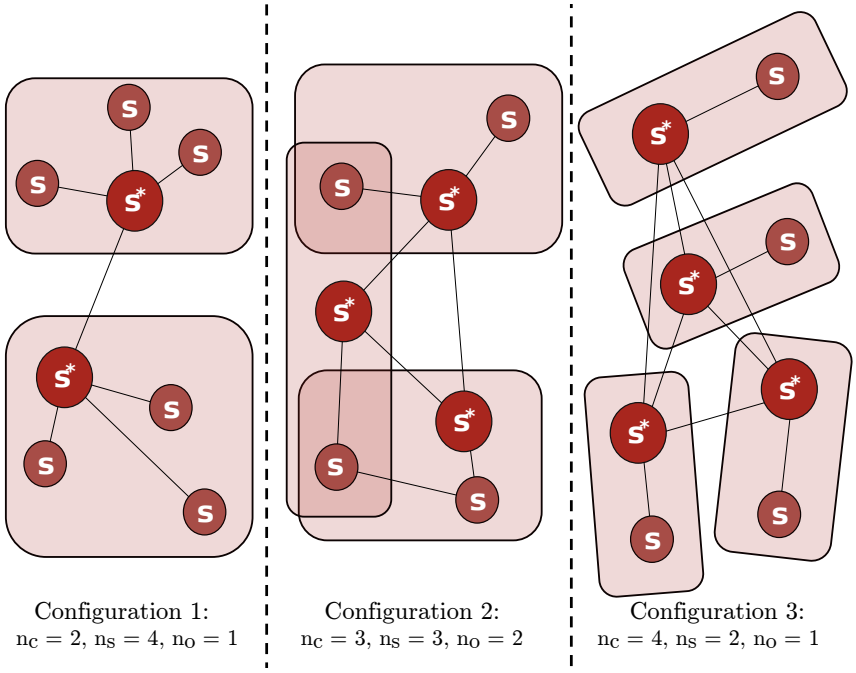


Figure 5.2: Three different parameter configurations and examples of valid communities. n_c specifies the number of communities, n_s the number of sensors in each community, and n_o the maximum number of overlapping sensors.

nicate models rather than send features. In contrast to the size of the features sent, we assume that the size of a model is negligible.

SCALABILITY Small values of n_s imply that the system is more scalable as communities need to share and analyze less data. Increasing n_s also increases the number of channels of communication and the amount of data that needs to be transferred to a community head. Therefore, the required computational capabilities of a system are inversely proportional to n_s.

5.4.3.2 *Number of Communities*

ACCURACY The total number of communities n_c affects the total number of datasets and learned models. Therefore, it has an im-

pact on the performance of an ensembled predictor. The larger n_c is, the more models are available and the more accurate the ensemble of models can be. When $n_c = 1$, a single community is formed and is equivalent to the case where $n_s = \|S\|$. In contrast, when $n_c = \|S\|$ and $n_o = 1$, sensors are their own community and no information sharing takes place. n_c and n_s are inversely proportional. n_c is bounded according to $1 \leqslant n_c \leqslant \|S\|$.

OVERHEAD We express the overhead imposed by n_c as the number of connections between sensors and their community heads. This overhead calculation is isomorphic to the overhead calculation of n_s and can be expressed as $n_c \cdot (n_s - 1)$.

SCALABILITY The parameter n_c affects the scalability of the system only in combination with n_s. We consider that scalability is only affected by the amount of data that needs to be collected and processed. For example, large values of n_c and low values of n_s create constellations where many communities need to process small amounts of data. The number of communities n_c alone does not negatively affect scalability on the chosen abstraction level.

5.4.3.3 *Maximum Sensor Overlap*

ACCURACY The parameter n_o corresponds to the maximum number of times a sensor can appear in different communities. Note that a sensor cannot appear twice in the same community, i. e., it is not allowed to send repeated information to a community head. As this parameter increases, more data is repeated among communities and more accurate models can be built. The bounds of n_o are defined by $1 \leqslant n_o \leqslant n_c$.

OVERHEAD As the parameter n_o is raised, communication overhead increases as sensors need to send their extracted features to multiple community heads. The increase in overhead introduced by n_o is simply calculated as $(n_o - 1)$. This parameter, however, directly influences the number of sensors within a community n_s. Furthermore, the larger n_o is, the more communities n_c can be accommodated. More overlap implied more overhead.

SCALABILITY On its own, the sensor overlap n_o does not influence the scalability of the system. However, it enables the establishment of more communities and, therefore, indirectly affects scalability through n_c.

5.4.4 *Community Formation*

We say that communities $\mathbb{C} = \{C_1, C_2, \ldots, C_n\}$ honor the set of pa- *honor parameters*

rameters n_s, n_c and n_o when $\mathbb{C}$ meets the conditions

$$\begin{aligned} \|C_i\| &= n_s, \\ \|\mathbb{C}\| &= n_c, \\ \left\|C_i \cap C_j\right\| &= n_o \quad \forall C_i, C_j \in \mathbb{C} \text{ and } i \neq j. \end{aligned}$$

The set of sensors S can be used in numerous ways to form communities $\mathbb{C}$, i. e., $C_i \subseteq S$, such that the predefined parameters n_s, n_c and n_o are honored. A unique solution does not necessarily exist.

The criteria by which sensors are grouped into communities is analogous to choosing the properties of an ensemble of models. Each community $C \in \mathbb{C}$ creating a model using the features provided by its member sensors. Therefore, the number of communities n_c specifies how many models are ensembled. Sensors can appear in multiple communities to replicate their features into different datasets. This is analogous to the sampling process boosting uses (see Section 5.2.2.1) to create datasets that have different feature distributions. The number of sensors in a community n_s determine the size and completeness of the dataset the community will use for training purposes. Finally, n_o introduces a factor that limits how many times features can be replicated into different datasets.

5.4.5 *Sensor Grouping Algorithms*

In this section, we present two algorithms to stochastically group sensors S into communities $\mathbb{C}$ that honor one or more of the parameters n_s, n_c or n_o. While n_c is a parameter that applies globally, parameters n_s and n_o may be the same or different for each community. With Algorithm 5.1, we present an algorithm that fixes n_s and varies n_c and n_o. The purpose of the algorithm is to create multiple communities with the same number of sensors, even if sensors need to be repeated among communities. Algocf 5.2 tries to do the opposite: it fixes n_o and n_c (whenever it is possible to do so), and leaves n_s to vary. This algorithm creates community constellations that allows us to easily test the impact of overlapping sensors.

ALGORITHM 1 Given the set of sensors S and the number of sensors in a community n_s, Algorithm 5.1 outputs n_c communities, i. e., $\|\mathbb{C}\| = n_c$, where each community $C \in \mathbb{C}$ has the same number of sensors n_s. The communities may also have different numbers of overlapping sensors n_o. The algorithm consists of two phases. In the first phase (lines 2 to 5), the algorithm selects a sensor that does not yet belong to another community to start a new community. The set Temp is used to keep track of the sensors that already belong to a community. The second phase of the algorithm (lines 6 to 9) randomly adds a sensor to C from the set difference $(S - C)$ until $\|C\| = n_s$.

```
input  : set of sensors S
input  : number of sensors n_s in a community
output : set of communities ℂ
 1 ℂ ← {∅}, Temp ← {∅}
 2 for s ∈ S do
 3     if s ∉ Temp then
 4         C ← {s}
 5         Temp ← Temp ∪ {s}
 6         while ‖C‖ < n_s do
 7             s ← rand(S − C)
 8             C ← C ∪ {s}
 9             Temp ← Temp ∪ {s}
10         ℂ ← ℂ ∪ {C}
11 return ℂ
```

Algorithm 5.1: Given a set of sensors S and a fixed parameter n_s, construct a set of communities $\mathbb{C}$ that honors the parameter n_s (number of sensors) and lets each community have different values of n_c (number of communities) and n_o (sensor overlap).

ALGORITHM 2 With a set of sensors S and the n_c and n_o parameters fixed, Algorithm 5.2 creates a set of communities $\mathbb{C}$ where $\|\mathbb{C}\| = n_c$ and no sensor overlaps more than n_o times among all communities. This algorithm creates communities containing different numbers of sensors n_s. The strategy of Algorithm 5.2 is as follows. First, lines 3 and 4 initialize the set $\mathbb{C}$ with n_c empty communities. The top-most loop (line 5) iterates over each sensor $s \in S$. The inner-most loop (line 8) distributes the sensors following a uniform distribution, i. e., $\mathcal{U}(1, n_o)$, into different communities. If communities from the set $\mathbb{C}$ finish empty, they are discarded.

5.4.6 *Community-based Collaborative Intrusion Detection*

After grouping sensors into communities following one of the previously presented algorithms, sensors can collaborate to detect intrusions in a network. A community of sensors $C_i \in \mathbb{C}$ can be interpreted as an overlay network that connects all its members $s \in C_i$ to its community head $s_i^\star$. Sensors are assumed to analyze the network traffic they monitor, extract features and forward the features to their community heads. Community heads are also assumed to be capable of communicating with each other to share intrusion detection models.

Each community head builds a ML model using an *aggregated training dataset* composed of all the features sent by the sensors directly connected to the community head. Community heads are also responsible for sharing their models with other community heads. We propose an efficient sharing mechanism in Chapter 6. In this work,

aggregated training dataset

```
input  : set of sensors S
input  : number of communities n_c
input  : maximum number of sensor overlap n_o
output : set of communities ℂ
 1 if n_o > n_c then
 2     n_o = n_c
 3 C_1, C_2, ..., C_{n_c} ← {∅}, {∅}, ..., {∅}
 4 ℂ ← {C_1, C_2, ..., C_{n_c}}
 5 for s ∈ S do
 6     x ← uniform(1, n_o)
 7     Temp ← {∅}
 8     for 1 to x do
 9         C ← rand(ℂ − Temp)
10         C ← C ∪ {s}
11         Temp ← Temp ∪ {C}
12 return ℂ
```

Algorithm 5.2: Given a set of sensors S and fixed parameters n_c (number of communities) and n_o (overlapping sensors), construct a set of communities $\mathbb{C}$ that honors the two fixed parameters. The number of sensors in communities n_s is not fixed and can vary for each community.

community heads build normality models for an anomaly-based intrusion detection system using LERAD. However, our proposed methodology is not bound by LERAD and can easily accommodate other ML techniques.

Having received the normality models created by each community, community heads create an ensemble of models to detect anomalies in the network traffic monitored by their community. Sensors within a community keep sending the features they extract to their community heads. Community heads then build an *aggregated testing dataset* and use LERAD to detect anomalies within this dataset. The generated alarms are sent to a central location where they can be sorted, aggregated and correlated. We randomly choose a community head to act as this central location.

aggregated testing dataset

5.5 EVALUATION

We present the results of testing our proposed community-based CIDS using LERAD as anomaly detector on top of a modified DARPA 99 dataset. We choose these two as they are known to perform well together. DARPA provides labeled data and, despite its deficiencies, does not hamper testing our methodology. This is because our objective is to determine how communities compare to a fully centralized and fully distributed system (with no collaboration). This objective does

not depend on having a realistic or error-free dataset, i. e., with the same dataset we compare different systems under similar conditions.

The next tests compare the capabilities of centralized, isolated and community-based CIDSs. Community-based CIDSs are generalizations of centralized and isolated systems that approach the performance of one or the other depending on how communication overhead and detection accuracy is traded. We acknowledge that a centralized CIDS should have the best detection performance while the opposite is true for a fully isolated CIDS. Our evaluations demonstrate how a community-based CIDS outperforms an isolated CIDS. Communities can perform as well as a centralized CIDS if communication overhead is disregarded.

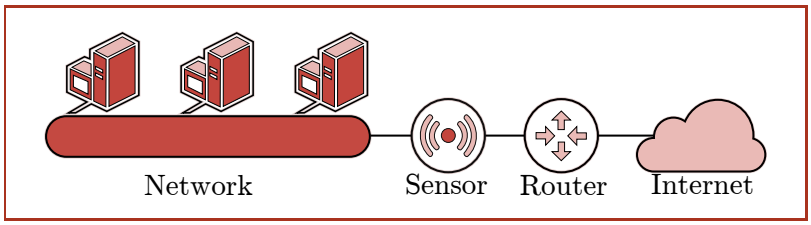
5.5.1 *Modifications to the DARPA 99 Dataset*

Our tests use a modified DARPA 99 dataset that reflects a scenario where multiple sensors are simulated to independently monitor parts of a network. For a full description of this dataset without our modifications, see Section 3.3.1. To construct aggregated training datasets, we use the third week of the incoming training traffic of DARPA which does not contain attacks[2]. For the aggregated testing datasets, we use the two available weeks of incoming testing traffic of DARPA. The testing dataset consists of normal traffic and 201 different attacks. Due to our modifications described next, 19 attacks are removed, i. e., 19 attacks are treated as if no sensor observed them.

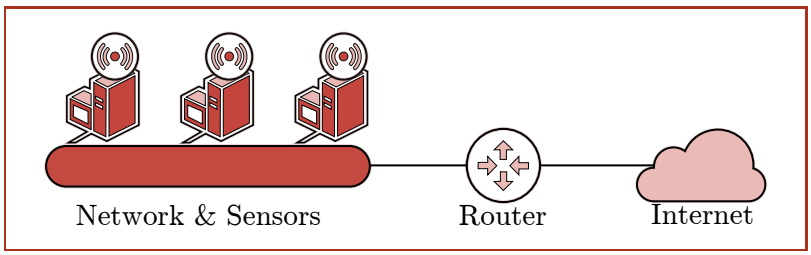
Figure 5.3 shows the original and our modified DARPA 99 capturing architecture. The DARPA 99 dataset was captured using a single sensor at the ingress point of external traffic, as shown in Figure 5.3a. We modified the dataset to reflect the architecture shown in Figure 5.3b. With our modifications, we simulate that each local host captures network traffic independent of each other.

Our modifications effected two changes in the resulting dataset (in contrast to the original DARPA 99 dataset). First, the modifications removed all packets from the dataset with a destination IP address of a host that does not exist in the local network. Packets discarded this way relate mostly to port scans and DoS attacks. Second, the modifications removed packets in the testing dataset that target hosts that do not appear in the training dataset. This is required as we assume that all sensors (hosts) are present, both, during training and testing of the anomaly detection models. Would this assumption not be made, many sensors during testing would not have an associated anomaly model. The resulting dataset consists of 15 sensors (one for each local host still in the dataset after our two modifications).

2 The DARPA dataset was created with synthetic traffic. The first three weeks of incoming traffic contain only normal traffic.



(a) Original DARPA 99 architecture: Outgoing network traffic is captured by one sensor that lies before the last end point before traffic reaches the Internet.



(b) Modified DARPA 99 architecture: Instead of a single sensor, each host captures its own outgoing network traffic.

Figure 5.3: The original and our modified architecture of the DARPA 99 dataset.

5.5.2 *Using LERAD in the Communities*

Every community head $s_i^\star$ uses LERAD to build a model of normality. Community heads are responsible for running LERAD on their aggregated training dataset to create conditional rules that model the normal behavior of network traffic within a community. These normality models are then shared with all other community heads. At each community head, an ensemble of models is built with all models, effectively recreating the same ensemble in each community head. Community heads use the ensemble model, which consists of collections of conditional rules, to test their aggregated testing dataset. Rule violations, or alarms, are sent to a central location where alarms can be sorted and reported. We choose a random community head to act as this central location.

Sensors extract 23 different features from the local network traffic they monitor. Therefore, the aggregated training and testing datasets consist of data points with 23 features. The features we select are known to work well with LERAD [Matthew V Mahoney and P. Chan, 2003]. For each observed network flow (see Section 4.2.1), we extract the date and time; the destination and source address; the destination and source port; the duration of the flow; the TCP flags of the first, second to last and last packets of the flow; the sum of payload sizes of packets in the flow; and the first eight words of flow.

5.5.3 *Experimental Setup*

We use recall and precision to evaluate our results. In this context, recall is the number of attacks detected over the total number of attacks. Precision is the true alarms, i. e., attacks detected, over the total number of alarms (true alarms + false alarms). Each experiment is run 500 times and the recall and precision are averaged over all runs. We do not include confidence intervals in our results as these are small and do not add useful information to the plots.

To measure recall and precision, we use the alarms issued by every community head. Recall that all community heads send their alarms to a predefined community head acting as a central location. Duplicated alarms within a 60 second interval are removed. We analyze alarms from high to low anomaly scores and determine if they correspond to a true or false positive. We stop the analysis process after a certain number of false alarms are processed. All remaining alarms are discarded and results are calculated.

Our experiments test the recall and precision of different community configurations. We explicitly distinguish between three important community configurations based on the number of sensors communities have:

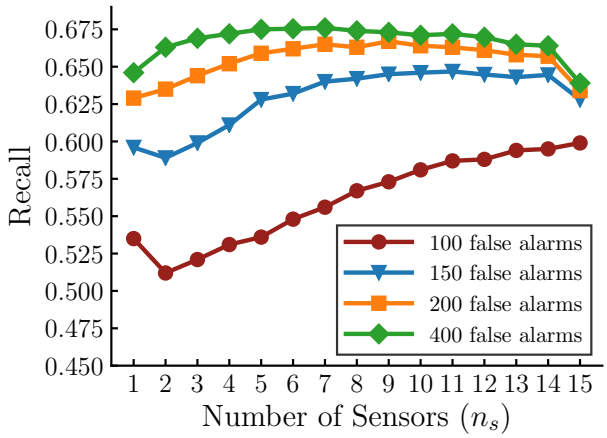
- Centralized Configuration ($n_s = \|S\|$ and $n_o = 1$). Only one community exists and all sensors send features to one community head. *centralized configuration*
- Isolated Configuration ($n_s = 1$ and $n_o = 1$). One community exists for each sensor for a total of $\|S\|$ communities. No collaboration between sensors is possible. *isolated configuration*
- Community Configuration ($n_s = x$ where $1 < x < \|S\|$). Communities are formed that enable different degrees of collaboration between sensors. *community configurations*

In our experiments, we expect the centralized configuration to outperform all others as all available data is used to create a single model. On the contrary, the isolated configuration is not expected to perform well as only weak models are ensembled. As communities include more sensors, their recall and precision should approach those of a centralized configuration. Due to the ensemble of models, we further estimate that certain community configurations could outperform a centralized one.

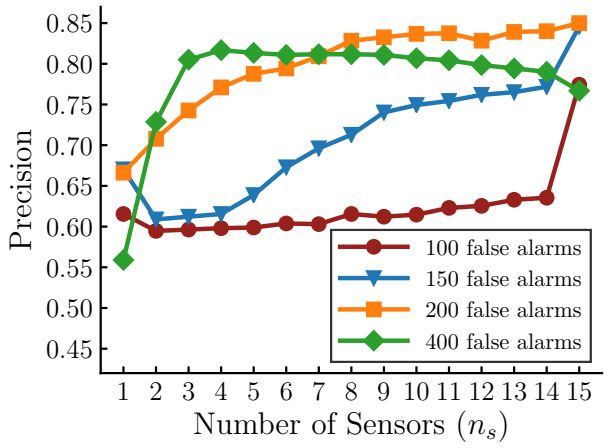
5.5.4 *Experimental Results*

Figure 5.4 shows a comparison of the recall and precision of every possible community size, as created using Algorithm 5.1, given as input the 15 sensors of our modified DARPA dataset. Figure 5.4 shows

the outcomes of our experiments when different numbers of false alarms are tolerated. For each experiment, we perform anomaly detection until the stated number of false alarms are issued. At that point, recall and precision are calculated with the processed alarms and the leftover alarms are ignored. High false alarm rates is one of the main weaknesses of anomaly detection. By discarding alarms, we restrict the false alarms to make the system usable in practice. We measure the detection capabilities of our communities using thresholds of 100, 150, 200 and 400 false alarms. The testing data consists of two weeks (10 days) of traces. Therefore, 100 false alarms correspond to an average of 10 false alarms per day, 150 to 15 false alarms, and so on.



(a) Recall after considering different number of false alarms.



(b) Detection precision after considering different number of false alarms.

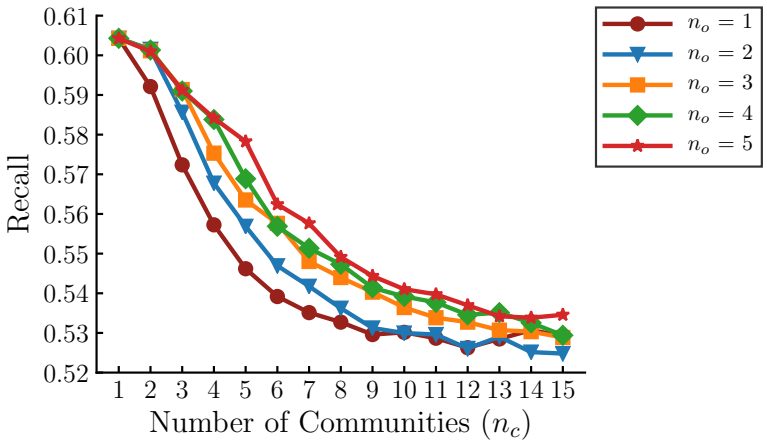
Figure 5.4: Recall and precision when communities have different number of sensors n_s. With each configuration, a different number of total false alarms are tolerated before dropping all other alarms.

After encountering, e. g., 100 false alarms, we report the recall and precision of our community-based anomaly detection methodology. In Figure 5.4b, we show that when communities grow in size, precision is improved. This corroborates our hypothesis that a centralized configuration would have the highest precision. As seen in Figure 5.4a and 5.4b, if the false alarms threshold is increased, some community sizes improve the recall in contrast to a centralized configuration, i. e., $n_s = 15$. With a threshold of 200 false alarms, most community sizes have better recall than the centralized configuration. Furthermore, with a higher threshold, the detection precision of all community sizes converges to the precision of the centralized configuration. With a threshold of 400 false alarms, every community outperforms, in terms of recall, the isolated community configuration as well as the centralized configuration. Above 400 false alarms, no significant changes are observed. However, as seen in Figure 5.4b, the precision drops as the false alarms increase. With the 200 false alarms limitation, community configuration where $n_s \in [9, 11]$ approach the precision of the centralized configuration.

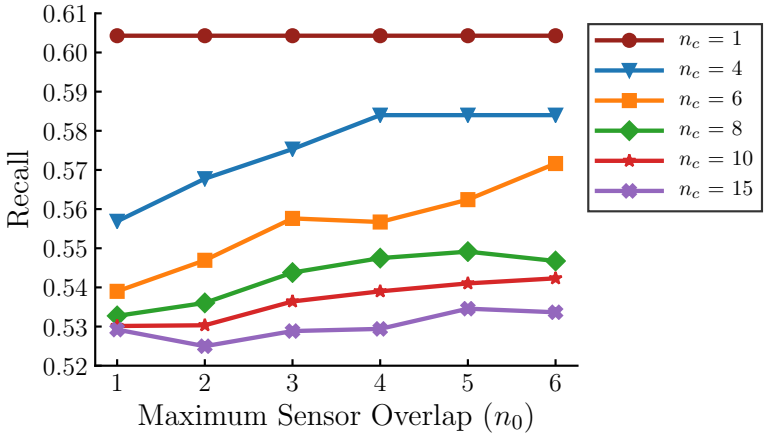
The maximum sensor overlap n_o has noteworthy properties that reflect on the recall of communities of fixed size. In Figure 5.5a, we illustrate the recall of communities of fixed size n_s and varying sensor overlap n_o. Recall generally improves with more sensor overlap n_o no matter the configuration. A centralized configuration, however, outperforms all others as expected. A centralized configuration with a sensor overlap of n_o is the same as creating n_o models with the features of every sensor and making an ensemble of all models. For this experiments, we created communities using Algorithm 5.2.

In Figure 5.5b, we show an isomorphic perspective of Figure 5.5a. This perspective, however, also illustrates that as the number of communities n_c increases, the impact of n_o is only slightly noticeable. The sensor overlap, therefore, is not as impactful as we expected.

Our results show that communities easily outperform isolated configurations and converge towards a centralized configuration. When the threshold of false alarms is raised, some community configurations can outperform a centralized configuration. This is due to the nature of ensemble learning which can generally improve the performance of weaker learners. Communities can improve the detection and precision ratio of collaborating sensors while lowering communication overhead as compared to a centralized configuration. Our results indicate that, for the particular case of using the DARPA dataset, we can find a combination of the parameters n_s, n_c, and n_o; plus a suitable false alarm threshold that enables community configurations to perform better than the centralized configuration, i. e., $n_s = 9, n_c = 4, n_o = 3$ and a false alarm threshold of 200. This is a particular setting that does not necessarily generalize to other datasets.



(a) Recall depending on the number of communities n_c and the maximum allowed sensor overlap n_o.



(b) Recall depending on the maximum sensor overlap n_o.

Figure 5.5: Recall when we vary the maximum allowed sensor overlap n_o. With Algorithm 5.2, we create communities that all have the same parameter n_o for a given community size n_c.

With different datasets, the parameters for the best performing system will certainly vary.

5.6 CONCLUSION AND LESSONS LEARNED

The intrusion detection community has many well established intrusion detection mechanism. However, with the advent of large networks, collaborative attackers and CIDSs as countermeasures, many mechanisms became outdated due to their centralized nature. We experience this problem, for example, when trying to use the LERAD algorithm with a day of the MAWI dataset (see Section 3.3.1). It is not uncommon for a day in the MAWI dataset to be larger than 10GiB in size. This is taking into account that packet payloads are not available

and only 15 minutes of traffic are recorded per day. LERAD could not be implemented to operate at the scale of the MAWI dataset without modifications.

Instead of modifying LERAD, we arrived at the conclusion that it is more viable to work on partitions of a dataset instead of modifying every centralized algorithm to fit a distributed environment. In live networks, however, partitioning network traffic is not straightforward. In a distributed scenario, sensors monitor network segments and do not know in foresight the amount of traffic they will observe. Therefore, we need to create groups of sensors that can share information with themselves to build models. This alone does not solve the issue of utilizing an inherently centralized mechanism; we still require a centralized component where data can be collected and learned from. We proposed the idea of communities to tackle these issues.

This chapter studied the possibility of using centralized intrusion detection mechanisms inside distributed CIDSs using communities. A community is a grouping of CIDS sensors that operates as a centralized system independent of other communities. All sensors within a community are responsible for extracting features from the network segments they monitor (without overlap) and sending the features to a community head. Each community appoints a sensor as community head which is responsible, besides receiving all features, for creating ML models. Community heads then share their models with each other to create ensembles of models. Sharing models has a significantly smaller communication overhead that sharing data and can be disregarded in the calculation of network overhead. When a community head needs to detect intrusions, it uses the ensemble of models (which is the same in every community head).

5.6.1 *Future Work*

In this work, we establish communities with stochastic algorithms and do not take into account the roles of the sensors. We acknowledge, however, that more intelligent strategies can be used to establish communities: not all sensors are expected to observe the same network traffic. A community head might be capable of creating better models when all traffic sent to it is similar (or different depending of the scenario). For example, all web servers might want to send their extracted network features to a common community head so that the community head can create better models of normality that describe web traffic.

The algorithms we propose to form communities are centralized. In future work, a system that implements the concept of communities would benefit when these community formation algorithms would work in a distributed environment.

5.6.2 *Chapter Summary*

This chapter began introducing the need of applying centralized intrusion detection systems within distributed CIDS. Communities were proposed as a mechanism that achieves this. Each community independently creates a ML model using the data generated by its members and shares the model with all other communities to establish an ensemble of models. Communities determine if network traffic contains intrusions by querying each model in the ensemble and choosing the answer given by the model with the most confidence.

Communities have three inherent parameters that have a direct impact in the communication overhead and the detection accuracy of their created ensemble of models. The parameters are the size of a community, the number of total communities and the maximum overlap of sensors within communities. These three parameters constitute the means by which we can form communities. We propose two different algorithms that stochastically form communities respecting some combination of parameters.

The first algorithm we propose focuses on establishing communities with a fixed size, i. e., the number of sensors that make up a community is the same for all communities. The first algorithm varies the number of communities and the overlap of sensors to accommodate communities with a fixed size. In contrast, the second algorithm we propose forms a fixed number of communities that all have a specific sensor overlap. It does so by varying the number of sensors within communities. These two algorithms are used to evaluate the intrusion detection capabilities of communities.

In the evaluation section of this chapter, we demonstrated how our community concept can be applied to detect intrusions in large networks using an inherently centralized mechanism within a distributed CIDS. We analyzed the particular case of implementing an anomaly-based intrusion detection mechanism where each community was responsible for creating a model of normality. The experiments used a modified version of the DARPA 99 dataset that reflects a scenario of multiple sensors. We concluded that communities can leverage communication overhead and detection accuracy to approximate the detection accuracy of a centralized system, i. e., a system that possesses all available information. Our experiments also showed that with certain parameter settings communities were able to outperform a centralized system. This was due to the creation of ensembles that theoretically improve upon the performance of single weaker models.

6

INTRUSION INFORMATION DISSEMINATION

CONTEXT

In the previous chapter, we proposed concepts for grouping sensors into communities so that CIDSs could use centralized mechanisms in distributed environments. Disseminating information was at the core of detecting network-wide anomalies, yet we used a tacit dissemination strategy. In fact, not only our work, but also most research into CIDSs assumes or takes for granted the process by which information is disseminated [Vasilomanolakis, Karuppayah, et al., 2015]. Instead of proposing adequate dissemination strategies, researchers often rely on central servers to collect information or on sub-optimal dissemination techniques like flooding. These inadequate strategies impose heavy restrictions on the size of communication networks and bring new sets of problems. A viable dissemination strategy that generalizes to different CIDSs is in demand. Dissemination strategies are of special importance if distributed system are to work not only in a theoretical but also in a practical sense.

In this chapter, we propose a viable strategy to efficiently disseminate intrusion information taking into account the requirements imposed by CIDSs. We further propose a set of requirements to make dissemination strategies practical and generalizable to most CIDSs.

A chapter overview is shown in Figure 6.1. This chapter constitutes the fourth contribution of this thesis. From the perspective of the CIDS architecture model we reference (see Section 2.3.3), the contribution covers the *Data Dissemination*, and *Data Correlation and Aggregation* layers. In the presentation of the contribution, two points of view are taken. From the point of view of the *Data Dissemination* layer, we propose a stochastic strategy specifically designed for CIDSs to efficiently disseminate information. To fully disseminate information to every collaborating member, our experiments showed that our strategy needs less than 20 percent of the messages sent by flooding. The strategy further enables the members of a CIDS to make deductions concerning information that has not yet been explicitly shared. This enables the members of a CIDS to make decision without needing to wait for all information to be disseminated. From the perspective of the *Data Correlation and Aggregation* layer, the proposed strategy

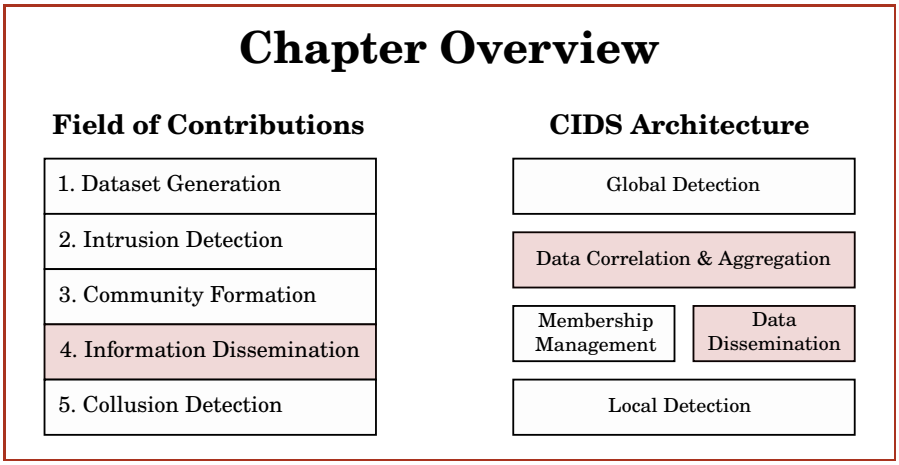


Figure 6.1: This chapter constitutes the fourth contribution of this thesis: *Information Dissemination*. This contribution is tied to the highlighted layers of our referenced CIDS architecture: *Data Dissemination* and, *Data Correlation and Aggregation*.

compresses and encodes information into constant-sized messages of sub-linear size[1].

This chapter is structured as follows. Section 6.1 introduces and motivates the need for a general purpose information dissemination strategy tailored towards CIDSs. We present specialized background, relevant for comprehending our dissemination strategy, in Section 6.2. Related work follows in Section 6.3 describing what CIDSs do to disseminate information. Our dissemination strategy is structured, to ease its comprehension, into three components. We give an overview of each component in Section 6.4. The first component, which relates to feature processing, is detailed in Section 6.5. We then follow with an explanation of the similarity deduction component in Section 6.6. In Section 6.7, we elaborate on the last component which deals with information dissemination. With the evaluation presented in Section 6.8, we demonstrate that our dissemination mechanism performs better and has advantages in comparison to the most commonly used dissemination mechanism. Finally, we conclude with our lessons learned, a view into the future and a chapter summary in Section 6.9.

6.1 INTRODUCTION

CIDSs rely on several NIDS to monitor large networks. NIDSs are responsible for observing and collecting information from network segments. A full network overview is then acquired when NIDSs exchange their information with each other. In distributed CIDSs, where

1 A data structure is sub-linear in size if it occupies less units of space than the total units of space occupied by all data items it stores.

there is no centralized information repository, NIDSs are expected to directly distribute information among themselves. Information distribution and dissemination, therefore, becomes the basis by which collaboration is established.

Correlating and aggregating shared information is at the core of establishing collaboration in a CIDS [Vasilomanolakis, Karuppayah, et al., 2015]. NIDSs distribute information, e. g., sufficient statistics [Lazarevic, Nisheeth Srivastava, et al., 2009], to enable others to correlate observations. Through correlations, NIDSs find common patterns that enable them to find network-wide intrusions. To distribute their findings, NIDSs use aggregation techniques to summarize what they and others observe into messages. Aggregation minimizes the footprints of the messages by making them as concise and informative as possible. Messages, however, still need to be disseminated intelligently to minimize the communication overhead of the system a whole.

6.1.1 *Problem Statement*

We tackle the problem of efficiently disseminating messages within NIDSs that would enable them to find distributed threats. Finding threats is achieved by correlating, aggregating and disseminating information using a strategy that minimizes communication overhead. Overhead is minimized by giving CIDS members the abilities to share incomplete information and to deduce what is missing from it. We rely on two tools to bestow CIDS members these abilities. First, we use the *Sketch* PDS to correlate and aggregate local and remote information to create messages suitable for dissemination. Second, Bayesian Networks are used to deduce the information hidden behind the aggregation process.

To create suitable messages for dissemination, we need to minimize the size of the messages and the number of times messages hop between hosts. Despite these restrictions, we set the goal of enabling every member to send their information to all others. If messages were naively sent from one member to every other, a total of $n \times (n-1)$ messages would need to be sent, for n members. The quadratic complexity of such a strategy would make the communication overhead pronounced. Alternatively, we could append information to a received message before forwarding the message to other members. This reduces the number of messages sent, but increases the size of the messages. Trading lower number of sent messages for bigger messages, effectively incurs in the same communication overhead. A solution to this problem needs, therefore, to minimize the number of sent messages while maintaining the size of the messages constant at most.

Consider the scenario presented in Figure 6.2. Five CIDS members wish to determine which other members have observed similar network patterns. Table 6.1 summarizes the knowledge members acquire after the shown messages are exchanged. For example, B knows how similar its observations are to A, as indicated by a check mark (✓). On the other hand, A does not know anything about B, as indicated by a question mark. If E wanted to know something about A, A would need to send a message to B and B would need to forward this message to E. Given an inefficient information dissemination technique, 16 additional messages would need to be sent to change every question mark to a check mark.

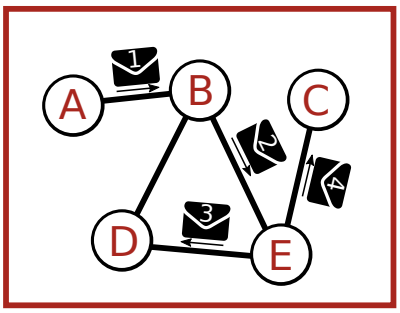


Figure 6.2: Five NIDSs members restricted by an underlay reagarding how messages can be shared.

	A	B	C	D	E
A		?	?	?	?
B	✓		?	?	?
C	?	?		?	✓
D	?	?	?		✓
E	?	✓	?	?	

Table 6.1: Illustration of what information each member knows of the other members after four messages are sent.

We wish to address the problem of distributedly filling out Table 6.1 with check marks, for each member, as efficiently as possible. Note that each member has a copy of this table that must be filled out. In filling out all tables, we are constrained to only sending small, constant-sized messages; thereby, minimizing the communication overhead. We create constant-sized messages by aggregating the information of different members into Sketches of fixed size. The caveat is, however, that Sketches cannot keep track of what belongs to whom. We study and design Bayesian Networks capable of deducing the members to whom aggregated information is associated.

6.1.2 *Challenges*

requirement categories

Information dissemination is a difficult task subjected to multiple requirements that related work tends to omit (see below). We classify the requirements related to the dissemination of information into two categories. The *general and practical* category of requirements make information dissemination usable in practice. The *reasoning* category of requirements enable CIDS members to process information efficiently, taking uncertainty into account.

While the first requirement category is inherent to the communication channel, the second category directly relates to how CIDSs handle information. Most CIDSs use Probabilistic Data Structures (PDSs), such as Bloom filters [Locasto et al., 2005; Vasilomanolakis, Krugl, et al., 2016], to perform aggregation and correlation. In such a context, the goal of these PDSs is to summarize *feature counts*. Feature counts are the histograms, or the number of times, certain contextual elements are seen in a predefined time window. In the context of network flows, examples of features counts are the number of times a specific IP address or a port number is seen in the last hour. In most modern NIDSs, feature counts hold the sufficient statistics needed to create intrusion detection models, e. g., [Lazarevic, Nisheeth Srivastava, et al., 2009; Nychis et al., 2008]. Many ML models solely require feature counts to recognize patterns, find clusters or retrieve information [Cha, 2007; Duda et al., 2012]. Therefore, dissemination strategies can be improved if they are tailored to the dissemination of feature counts.

feature counts

To summarize, the following are requirements needed to properly disseminate information within CIDSs:

- General and Practical Requirements

  general and practical requirements

  1 Efficient. CIDSs need to minimize the communication overhead by using small messages of constant size.

  2 Real-time. CIDSs need to process information and make decisions without much delay.

  3 Privacy-preserving. The messages disseminated within a CIDSs must not expose sensitive information of the source.

  4 Resilient. In this context, CIDSs should avoid creating a SPoF.

- Reasoning Requirements

  reasoning requirements

  5 Capable of Deductions. CIDS members should have the ability to reason accurately from incomplete or missing information.

  6 Focused on feature count dissemination. CIDSs should optimize the dissemination process to handle feature counts as these are in most cases sufficient to create intrusion detection models.

6.1.3 *Chapter Contributions*

In this chapter, we propose a dissemination strategy that finds CIDS members experiencing similar events. We take into account the previously defined *general and practical* requirements as well as the *reasoning* requirements to design the strategy. The proposed strategy, first, efficiently encodes, aggregates and correlates feature counts using Sketches. Second, the strategy trains Bayesian Networks to enable CIDS members to make similarity deductions from aggregated feature counts stored in multiple Sketches.

We evaluate our dissemination strategy and how CIDS members use it to infer similarities using realistic and synthetic data. Our experimental results show that when using 80 percent of the information used by flooding techniques, we can deduce feature similarities with small error margins. If 50 percent of the information is used instead, similarities can still be deduced better than the baselines, i. e., random or fixed similarity choices.

The proposed dissemination strategy is not only applicable within the context of CIDSs. It is a general strategy applicable to other problem spaces where it is useful to deduce similarities distributedly (taking communication overhead into account). Other potential domains include, but are not limited to, Software Defined Networking (SDN), edge computing and the placement of cache servers within Content Distribution Networks (CDNs).

6.2 SPECIALIZED BACKGROUND

This section introduces *Count-Min Sketches*, *Divergences* and *Bayesian Networks*. The dissemination strategy proposed herein combines these three topics to deduce similarities between observations made by CIDS members. A description of this problem is found in Section 6.1.1. *Count-Min Sketches* are used to encode and aggregate observations from different sources for distribution. *Similarity Metrics* define ways to compare distributed data. Finally, *Bayesian Networks* provide the probabilistic machinery needed to deduce similarities from aggregated observations.

6.2.1 *The Count-Min Sketch Probabilistic Data Structure*

One of the main purposes of PDSs is to track elements (in some way or form) using sub-linear space in relation to the size of the elements being tracked [Cormode and S. Muthukrishnan, 2005]. Many PDSs are used to compress, aggregate and distribute data. Their efficiency in doing so has earned them widespread use within network communications research.

Many PDSs exhibit specific advantages and disadvantages. Bloom filters, perhaps the most popular of these, have been used to detect collaborative attacks [Vasilomanolakis, Krugl, et al., 2016], route network resources [Fan et al., 2000] or create overlay networks [Kostić et al., 2003]. Classical Bloom filters, however, have certain disadvantages that make them unsuitable in our context due to our requirements (see Section 6.1.2). In order to achieve our goals, we want a PDS that can do two things: First, the PDS must be capable of tracking element counts. Second, the PDS must be able to calculate (or approximate) the distance between the element counts it encodes against the counts encoded by another PDS of the same type. Count-Min Sketches have

these capabilities [Anceaume et al., 2013; Cormode and M. Muthukrishnan, 2012]. From this point forward, we use the term *Sketch* to refer to a *Count-Min Sketch*.

Sketches

Sketches are PDSs able to efficiently count, quickly update and accurately approximate streams of items in sub-linear space[2] [Cormode and S. Muthukrishnan, 2005]. Given the element set

$$\mathbf{e} = \{e_1, e_2, \ldots, e_N\},$$

element count definition

where each element e_i has been observed a total of $f(e_i)$ times, considering $f : \mathbf{e} \to [0, \infty)$, a Sketch approximates the element count $f(e_i)$ with $\hat{f}(e_i)$. The approximation $\hat{f}(e_i)$ has probabilistic guarantees governed by user parameters.

Sketch formalization

We formally define a Sketch as $\mathcal{S}(\mathbf{M}, \mathbf{H})$. The set $\mathbf{M}$ consists of d vectors of w size, as in $\mathbf{M} = \{\mathbf{m}_1, \ldots, \mathbf{m}_d\}$, $\mathbf{m}_i = \{m_1, \ldots, m_w\}$. The set $\mathbf{H}$ is comprised of d pairwise-independent hash functions, as in $\mathbf{H} = \{h_1, \ldots, h_d\}$, $h_i : \mathbf{e} \to [1, w)$. The value of d and w can be manually specified or, if probabilistic guarantees are desired [Cormode and S. Muthukrishnan, 2005], derived from the parameters δ and ϵ as $d = \ln(1/\delta)$ and $w = \lceil 2/\epsilon \rceil$. Each element in $\mathbf{m}_i$ is initialized to zero, i. e., $\forall m \in \mathbf{m}_i : m = 0$. To simplify the notation, we use $\mathbf{M}(i, j) = m_j \in \mathbf{m}_i$.

The counts $f(e_i)$ are added to a Sketch using the update rule

$$\mathbf{M}(j, h_j(e_i)) = \mathbf{M}(j, h_j(e_i)) + f(e_i); \quad \forall j \in \{1, \ldots, d\}.$$

We obtain the approximated counts $\hat{f}(e_i)$ by computing the smallest hashed value of the element as

$$\hat{f}(e_i) = \arg\min_j \ \mathbf{M}(j, h_j(e_i)).$$

6.2.2 *Divergences of Sketches*

similarity metric

Similarity metrics compare the distance between two or more entities in a mathematical space. They effectively measure how objects differ from each other. Similarity metrics are commonly used for recognizing, retrieving, correlating and clustering information [Cha, 2007]. Given set Q, elements $x, y, z \in Q$ and distance function $d : Q \times Q \to \mathbb{R}$, the following properties are satisfied if d is a similarity metric [Anceaume et al., 2013]:

divergence

A distance function that measures the similarity of a Probability Mass Function (PMF) is known as a *divergence*. A divergence commonly satisfies a subset of the properties that define a similarity metric.

*Sketch *-metric*

We can compute the similarity of two Sketches using especially designed divergences [Anceaume et al., 2013; Cha, 2007]. The *Sketch *-metric*, proposed by Anceaume et al., is a divergence family that

2 A data structure with the sub-linear space property needs less than N units of space to save N items.

1. $d(x,y) \geqslant 0$ (non-negativity)
2. $d(x,y) = 0 \iff x = y$ (identity of indiscernibles)
3. $d(x,y) = d(y,x)$ (symmetry)
4. $d(x,y) + d(y,z) \geqslant d(x,z)$ (triangle inequality)

finds similarities between two Sketches. Given Sketches $\mathcal{S}^{(1)}(\mathbf{M}^{(1)}, \mathbf{H})$ and $\mathcal{S}^{(2)}(\mathbf{M}^{(2)}, \mathbf{H})$, encoding element sets $\mathbf{e}^{(1)}$ and $\mathbf{e}^{(2)}$, the *Sketch *-metric* approximates any of the divergences in the set $\mathcal{Q} = \{$ *Kullback-Leibler divergence, Jensen-Shannon divergence, Bhattacharyya distance* $\}$. Note that each Sketch is associated with a different set of vectors $\mathbf{M}^{(i)} = \{\mathbf{m}_1^{(i)}, \ldots, \mathbf{m}_d^{(i)}\}$ and each vector is associated with different values $\mathbf{m}_j^{(i)} = \{m_1^{(i,j)}, \ldots, m_w^{(i,j)}\}$. The set of hash functions $\mathbf{H}$, and parameters d and w are shared among all Sketches. We denote the sum of all element counts of a vector $\mathbf{m}_j^{(i)} \in \mathbf{M}^{(i)}$ as $\|\mathbf{m}_j^{(i)}\| = \sum_{k=1}^{w} m_k^{(i,j)}$.

divergence set $\mathcal{Q}$

sum of element counts $\|\mathbf{m}_j^{(i)}\|$

The *Sketch *-metric* is defined as:

$$\phi\left(\frac{\mathbf{e}^{(1)}}{\|\mathbf{e}^{(1)}\|}, \frac{\mathbf{e}^{(2)}}{\|\mathbf{e}^{(2)}\|}\right) \approx \arg\max_j \phi\left(\frac{\mathbf{m}_j^{(1)}}{\|\mathbf{m}_j^{(1)}\|}, \frac{\mathbf{m}_j^{(2)}}{\|\mathbf{m}_j^{(2)}\|}\right);\ \forall \phi \in \mathcal{Q} \tag{6.1}$$

The divergence between the PMFs of the element sets $\mathbf{e}^{(1)}$ and $\mathbf{e}^{(2)}$ (left of Equation 6.1) are approximated by finding the largest divergence between the PMFs of $\mathbf{m}_j \in \mathbf{M}^{(1)}$ and $\mathbf{m}_j \in \mathbf{M}^{(2)}$ (right of Equation 6.1). We use the shorthand $\phi(\mathcal{S}^{(1)}, \mathcal{S}^{(2)})$ to refer to the divergence between Sketches $\mathcal{S}^{(1)}$ and $\mathcal{S}^{(2)}$.

divergence of two Sketches $\phi(\mathcal{S}^{(1)}, \mathcal{S}^{(2)})$

6.2.3 *Bayesian Networks*

A Bayesian Network is a directed acyclic graph that represents the probabilistic influence that random variables exert upon each other. Formally, a Bayesian Network is defined as $\beta = (\mathcal{X}, \mathcal{W}, \theta)$ [Reed et al., 2014]. Random variables are represented by the nodes of the set $\mathcal{X}$. The directed edges of these nodes are captured by the set $\mathcal{W}$. The parameter set θ represents all the Conditional Probability Tables (CPTs) that encode the probabilistic relationships, encoded by the edges $\mathcal{W}$, of nodes in $\mathcal{X}$.

Bayesian Networks enable us to query for the probability of random variables given that we know the probability of some other ones. If we let some random variables in $\mathcal{X}$ be fixed to specific values, we can compute the posterior distribution of one or a set of the random variables in $\mathcal{X}$ which were not fixed. This is known as *inference* and is what enables us to deduce similarities within the members of a CIDS.

Bayesian Network inference

6.3 RELATED WORK

Sharing information is the key mechanism that enables CIDSs to effectively detect attacks in a distributed system. Most systems relying on an information distribution mechanism do not explicitly address the problem of minimizing communication messages and reducing network overhead. Wu et al. [2003] developed a CIDS that improved the performance of single IDSs by allowing IDSs to share events with each other. The events are processed with an inference engine that reduced false positives. Although different inference engines can be distributed among members of the CIDS, the authors do not explicitly address how events are disseminated within the network.

Oikonomou et al. [2006] proposed a collaborative overlay, known as *DefCOM*, that enables different NIDSs to collaboratively detect and mitigate DDoS attacks. Within the overlay, NIDSs send alert messages to each other to identify when bandwidth rates are expected to be exceeded. The authors state that "alarm messages are flooded on the overlay, in a controlled manner to suppress duplicate messages" [Oikonomou et al., 2006]. The flooding mechanism employed is inefficient and does not minimize network overhead.

Gamer [2012] proposed a collaborative anomaly-based detection system capable of detecting large-scale network attacks. Just as in the aforementioned systems, information exchange is the key to the detection capabilities of the system. This time, however, the author recognizes the network overhead issue and proposes a system that deals with it. The proposed solution restricts communication to neighbouring peers only. The system could be more efficient and less complex if this restriction would be relaxed while still minimizing the network overhead.

Many security systems mitigate network communication overhead by using PDSs to encode and distribute information. Bloom filters have been a popular tool to achieve this. In the work of Vasilomanolakis, Krugl, et al. [2016], the proposed system detects distributed attacks by disseminating Bloom filters among collaborating NIDSs. Bloom filters encode the IP addresses of network actors associated with malicious activity. With these Bloom filter encodings, some distributed attacks can be detected by computing the bits that overlap between two or more Bloom filters. The Bloom filters are distributed using a form of gossiping [Kermarrec et al., 2007]. Gossiping techniques do not guarantee that disseminated messages reach all communication members in an overlay. Instead, the messages reach all members with some probabilistic guarantees.

Yan et al. [2006] use Bloom filters to improve the performance of signature-based collaborative spam detection systems. In such systems, spam is detected by comparing unknown content against signatures spread among collaborating peers. Signature-based spam de-

tection is effective, albeit computationally expensive. Furthermore, in a distributed environment, the overhead of sharing signatures can be substantial. To improve the lookup, storage and merging of signatures, as well as reducing the communication overhead of synchronizing signatures, the authors use Bloom filters.

Boggs et al. [2011] presented a distributed NIDS that detects attacks by comparing network traffic against signatures of normal and abnormal traffic. The normal and abnormal signatures correspond to high order n-grams observed in the payloads of normal and abnormal packets, respectively. The n-gram analysis is based on the effective ANAGRAM algorithm [Wang et al., 2006]. The signatures are stored in Bloom filters and shared with collaborating members to distributedly identify attacks. The Bloom filters preserve the privacy of the analyzed traffic, minimize network overhead and reduce signature lookup times.

6.4 OVERVIEW OF THE DISSEMINATION STRATEGY

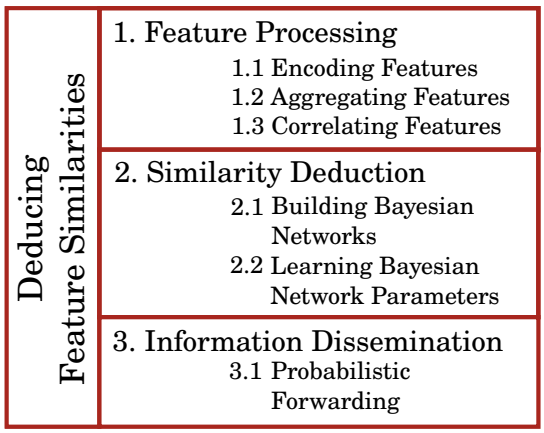


Figure 6.3: Components that build up our mechanism to deduce the feature similarities between CIDS members.

interpretation of similarity

Our ability to identify the similarity between CIDS members relies on them recording summaries of their observations in the form of feature counts. The interpretation of similarity is, therefore, directly related to the features being counted. For instance, if we use the malicious IP address of raised alarms as the feature to count, we would interpret that similar members are those experiencing attacks from the same malicious IP. Beyond detecting one-dimensional similarities, our dissemination strategy is able to tell different degrees of similarity when multiple features are taken into account.

1. feature processing component

2. similarity deduction component

The deduction of feature similarities is a process that uses three different components (see Figure 6.3). In the first component, called *Feature Processing*, feature counts are encoded, aggregated and correlated using Sketches as detailed in Section 6.5. In the *Similarity Deduction*

component, as detailed in Section 6.6, Bayesian Networks are build and trained to deduce similarities from Sketches encoding feature counts. Finally, in the *Information Dissemination* component explained in Section 6.7, we develop a stochastic message forwarding technique. The technique exploits Bayesian Networks and Sketches to efficiently distribute feature counts such that similarity deductions can be made.

3. information dissemination component

6.5 FEATURE PROCESSING: ENCODING COUNTS WITH SKETCHES

The first component needed to deduce feature similarities is that of encoding, aggregating and correlating feature counts. Sketches provide the means to achieve these. Sketches encode feature counts by storing these in the data structure itself. Aggregation is performed by summing Sketches together (see below). Correlation is then the process of estimating the difference (or similarity) between two Sketches using a divergence (Section 6.2.2).

1.1 encoding features

Feature counts encoded in Sketches can be interpreted as the histogram or Probability Density Function (PDF) of the counts. These PDFs are compared using the Sketch *-metric using one of different divergence functions. We use the Jensen-Shannon Divergence (JSD) [Lin, 1991] on top of the Sketch *-metric because of two of its properties. First, the JSD is symmetric with respect to its domain, making Sketch comparisons consistent. Second, the co-domain of the JSD is in the range $[0, 1]$ when base two logarithms are used. This co-domain enables bounded similarity comparisons.

1.2 aggregating features

sketch addition

Aggregating encoded feature counts is the process of adding two Sketches together. Given Sketches $\mathcal{S}^{(A)}(\mathbf{M}^{(A)}, \mathbf{H})$ and $\mathcal{S}^{(B)}(\mathbf{M}^{(B)}, \mathbf{H})$, Sketch addition is defined as

$$\begin{aligned} \mathcal{S}^{(A+B)} &= \mathcal{S}^{(A)} + \mathcal{S}^{(B)} \\ \mathcal{S}^{(A+B)}(\mathbf{M}^{(A+B)}, \mathbf{H}) &= \mathcal{S}^{(A)}(\mathbf{M}^{(A)}, \mathbf{H}) + \mathcal{S}^{(B)}(\mathbf{M}^{(B)}, \mathbf{H}) \\ \mathbf{M}^{(A+B)}(i, j) &= \mathbf{M}^{(A)}(i, j) + \mathbf{M}^{(B)}(i, j) \end{aligned}$$

Aggregated feature counts can be interpreted as adding histograms or mixing PMFs. If $\mathtt{PMF}^{(A)}$ and $\mathtt{PMF}^{(B)}$ are the PMFs represented by Sketch $\mathcal{S}^{(A)}$ and $\mathcal{S}^{(B)}$, respectively, then the PMF of $\mathcal{S}^{(A+B)}$ is $\mathtt{PMF}^{(A+B)} = 0.5 \cdot \mathtt{PMF}^{(A)} + 0.5 \cdot \mathtt{PMF}^{(B)}$. Aggregates alone are not enough to accurately calculate feature count similarities. However, these aggregates become sufficient when combined with our similarity deduction component (Section 6.6).

1.3 correlating features

Correlating feature counts is the process of calculating the divergence of the PMFs (and PMF mixtures) of the feature counts as encoded by Sketches. When using the Sketch *-metric with the JSD to compare Sketches, the JSD approximates the true divergence of the feature counts. JSD values close to zero indicate that two PMFs are similar, while values close to one indicate that they are different. In

Example 6.1, we illustrate the difference between calculating the divergence of PMFs or of Sketches encoding the PMFs.

Example 6.1: Feature Similarities

The PMFs of the observations made by three collaborators (A, B and C) may look like those shown in Figure 6.4. The divergences between these observations (and their aggregations) are shown in Table 6.2. Each table cell corresponds to the divergence between the observations (and aggregations) represented by the first column and row associated to the cell. The notation X + Y corresponds to the aggregation of the PMFs of X and Y. The top three rows show the divergences between PMFs. The bottom three show the divergences when comparing PMFs encoded in Sketches. Notice their similarity.

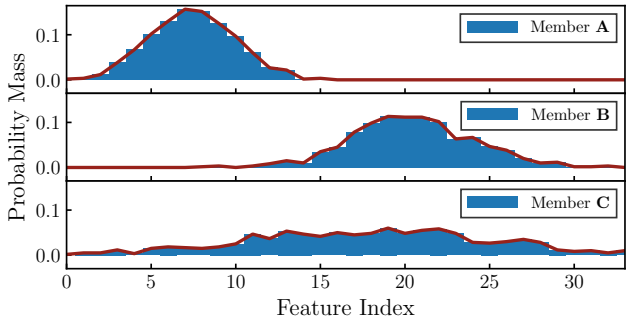


Figure 6.4: A scenario of three members, each with different PMFs, that describe the features they observed. Member A has mostly made observations from the fifth to the tenth feature. In contrast, member C has observed almost all features.

With the proposed techniques to process feature counts, calculating similarities between collaborators no longer requires the distribution of every count. Using Table 6.2 as an example, consider the scenario where B receives $\mathcal{S}^{(A)}$ and $\mathcal{S}^{(A+C)}$. From the divergence $\phi(\mathcal{S}^{(B)}, \mathcal{S}^{(A)}) = 0.91$ (from Table 6.2), B knows that it does not share many observations with A. B also knows $\phi(\mathcal{S}^{(B)}, \mathcal{S}^{(A+C)}) = 0.19$. Although B only has an aggregate that involves C, B can deduce that the gain in similarity from 0.91 to 0.19 must be due to a high similarity that exists between B and C, i. e., $\phi(\mathcal{S}^{(B)}, \mathcal{S}^{(C)}) = 0.21$. Thereafter, the similarity between B and C can be deduced to be medium-high without directly observing the feature counts of C. Although it is not obvious from this simple scenario, the benefits of this deduction strategy are apparent when aggregations involve multiple Sketches. The following section automates this deduction strategy when multiple aggregations are taken into account.

Divergence of PMFs

	A	B	C	A + B	A + C	B + C
			without Sketches			
A	0.00	0.95	0.56	0.30	0.46	0.71
B	0.95	0.00	0.22	0.29	0.21	0.09
C	0.56	0.22	0.00	0.14	0.12	0.04
			with Sketches			
A	0.00	0.91	0.51	0.29	0.45	0.65
B	0.91	0.00	0.21	0.27	0.19	0.08
C	0.51	0.21	0.00	0.12	0.11	0.04

Table 6.2: Divergence of the PMFs shown in Figure 6.4 for A, B, C and their aggregations. Each cell shows the divergence between the PMFs that belong to the members (or their aggregations) corresponding to the first column and row of the cell. The top three rows display the divergence between PMFs. The bottom three rows display the divergence between the same PMFs encoded with Sketches.

6.6 SIMILARITY DEDUCTION: USING BAYESIAN NETWORKS TO DEDUCE SIMILARITIES

The previous section presented the first component needed to deduce feature similarities. This first component used Sketches to encode, aggregate and correlate feature counts. This section presents the second component, that which deduces Sketch similarities. To automatically deduce similarities from Sketches, we use Bayesian Networks. We first tackle the task of designing a Bayesian Network capable of reasoning out, after some evidence, how Sketch similarities are related to each other. Secondly, we learn the parameters of the proposed Bayesian Network.

6.6.1 *Bayesian Networks for Deducing Similarities*

Designing a Bayesian Network $\beta = (\mathcal{X}, \mathcal{W}, \theta)$ (see Section 6.2.3) corresponds to defining the set of nodes (or random variables) $\mathcal{X}$ and the directed edges $\mathcal{W}$ that connect the nodes. The CPT parameters θ are either manually supplied or learned from data. We propose to model the nodes as discrete random variables that describe the discretized divergence $\phi(\mathcal{S}^{(X)}, \mathcal{S}^{(Y)})$ between the two Sketches $\mathcal{S}^{(X)}$ and $\mathcal{S}^{(Y)}$. The divergence $\phi(\mathcal{S}^{(X)}, \mathcal{S}^{(Y)})$ is calculated using $\phi = \text{JSD}$ as this facilitates the discretization process due to the JSD having a bounded co-domain [Lin, 1991].

We use a compact notation to reference the specific Sketch comparisons represented by nodes of our Bayesian Networks, the Sketches the nodes compare and the Sketch aggregations they include. Sketches of the form $\mathcal{S}^{(X)}$ are simply referred to as X. The Sketch aggregation $\mathcal{S}^{(X+Y)}$ is referred to as XY. To reference the divergence represented

node divergence notation

by a nodes, we use the notation $(X|Y) = \phi(\mathcal{S}^{(X)}, \mathcal{S}^{(Y)})$. We say that the node $(X|Y)$ contains the Sketches X and Y.

contents of a node

6.6.1.1 *The Nodes of the Bayesian Network*

The nodes $\mathcal{X}$ of our Bayesian Network are created taking into account two parameters $\mathbf{C}$, a set of Sketches, and n, the maximum number of Sketch aggregations. To aid us in the process of creating nodes, we define the set of all subsets of $\mathbf{C}$ as $\mathcal{P}(\mathbf{C})$ and the set of all combinations of $\mathbf{C}$ with k elements as $\mathcal{P}_k(\mathbf{C}) = \{\mathbf{X} \in \mathcal{P}(\mathbf{C}) : \|\mathbf{X}\| = k\}$.

parameters $\mathbf{C}$ and n

three step node creation approach

node candidates

We use a three step approach to create the nodes $\mathcal{X}$. First, we obtain the set of combinations of $\mathbf{C}$ with n elements or less $\mathbf{Q}_n(\mathbf{C}) = \{\mathcal{P}_1(\mathbf{C}) \cup \ldots \cup \mathcal{P}_n(\mathbf{C})\}$. Second, we create node candidates by forming element pairs using the elements of $\mathbf{Q}_n(\mathbf{C})$, as in $\mathbf{S} = \mathcal{P}_2(\mathbf{Q}_n(\mathbf{C}))$. This results in the set $\mathbf{S}$ having multiple elements $\mathbf{s} \in \mathbf{S}$. Each element $\mathbf{s}$ has two sets of Sketches, denoted $\{\mathbf{s}_1, \mathbf{s}_2\} = \mathbf{s}$, where $\mathbf{s}_i \subseteq \mathbf{C}$. Third, to finally create the nodes of the Bayesian Network, we eliminate the redundant elements of $\mathbf{S}$. The set $\mathbf{s}$ is redundant when it meets any of the conditions $\mathbf{s}_1 \subseteq \mathbf{s}_2$ or $\mathbf{s}_2 \subseteq \mathbf{s}_1$. The set of nodes $\mathcal{X}$ can now be formally defined as $\mathcal{X} = \{(\mathbf{s}_1|\mathbf{s}_2)\, \forall \mathbf{s} \in \mathbf{S} : \mathbf{s}_1 \not\subset \mathbf{s}_2,\ \mathbf{s}_2 \not\subset \mathbf{s}_1\}$, where we follow the notation we have introduced to reference the divergences represented by nodes. The number of elements in each of $\mathbf{s}_1$ and $\mathbf{s}_2$ is an important property of a node which we define as *rank*. Formally, the rank of the node $(\mathbf{s}_1|\mathbf{s}_2) \in \mathcal{X}$ is the sorted tuple

node rank

$$\text{rank}((\mathbf{s}_1|\mathbf{s}_2)) = \begin{cases} (\|\mathbf{s}_1\|, \|\mathbf{s}_2\|) & \text{if } \|\mathbf{s}_1\| \leqslant \|\mathbf{s}_2\| \\ (\|\mathbf{s}_2\|, \|\mathbf{s}_1\|) & \text{if } \|\mathbf{s}_2\| < \|\mathbf{s}_1\| \end{cases}, \tag{6.2}$$

where $\|\mathbf{s}_i\|$ refers to the number of aggregations in Sketch $\mathbf{s}_i$.

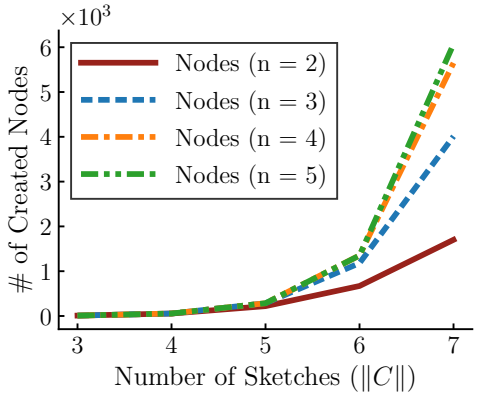


Figure 6.5: The number of nodes created for a Bayesian Network grow exponentially when different values of the parameters n and $\mathbf{C}$ are used.

As a result of our methodology, the number of nodes in $\mathcal{X}$ grows exponentially relative to the parameters n and $\mathbf{C}$. Figure 6.5 shows the exponential growth of the number of nodes in $\mathcal{X}$ as a function of n and $\mathbf{C}$. Without additional improvements, inference on a Bayesian Network becomes intractable when $\|\mathbf{C}\| > 6$ due to the number of

nodes. Sharing parameters between the nodes of the network, however, reduces the complexity of the network to such an extent that inference becomes tractable. This is the subject of Section 6.6.1.3. In Example 6.2, we show the results of the node creation process.

Example 6.2: Node Creation Process

Consider the nodes of the Bayesian Network in Figure 6.6 (ignoring the edges). The nodes are created by following the previous three step approach using three Sketches $\mathbf{C} = \{\mathcal{S}^{(A)}, \mathcal{S}^{(B)}, \mathcal{S}^{(C)}\} = \{A, B, C\}$ and a maximum number of aggregated Sketches $n = 2$. Notice the three types of nodes created and their rank. The top-most nodes compare unaggregated Sketches, e. g., $(A|C)$, and have a rank of $(1, 1)$. The middle nodes compare one unaggregated Sketch against an aggregated Sketch, e. g., $(A|BC)$, and have a rank of $(1, 2)$. The bottom-most nodes compare an aggregated Sketch against another aggregated one, e. g., $(AB|BC)$, and have a rank of $(2, 2)$.

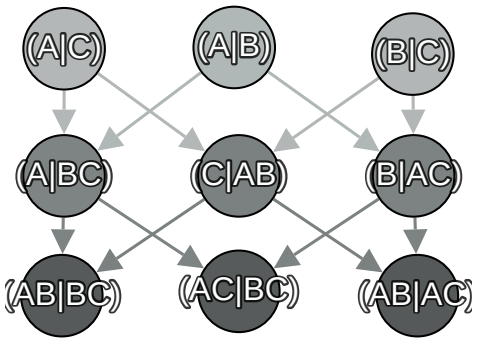


Figure 6.6: A small Bayesian Network designed to deduce the similarity of three Sketches. Each Sketch can have up to two aggregations.

6.6.1.2 *The Edges of the Bayesian Network*

Multiple methods can be used to model the probabilistic relationships between the nodes $\mathfrak{X}$ as encoded by the edges $\mathcal{W}$. To reduce the complexity of performing inference with a Bayesian Network, we aim at minimizing the number of parents for any given node while retaining the predictive capabilities of the Bayesian Network as modeled by the CPTs in θ. The biggest limiting factor for learning the θ parameters is the number of parents of a node. The larger the number of parents, the more data is needed to estimate the parameters. A straightforward method to minimize parents is to create directed edges between each node and higher ranked nodes that share any of the two Sketches they contain. This guarantees that all needed conditional independent relationships are taken into account. We use the term *simple edge creation method* to refer to this edge building proce-

simple edge creation method

dure and formalize it with the set of directed edges

$$\mathcal{W} = \left\{ \left((\mathbf{s}_1^{(1)} | \mathbf{s}_2^{(1)}), (\mathbf{s}_1^{(2)} | \mathbf{s}_2^{(2)}) \right) \middle| \begin{array}{l} rank\left((\mathbf{s}_1^{(1)} | \mathbf{s}_2^{(1)}) \right) < rank\left((\mathbf{s}_1^{(2)} | \mathbf{s}_2^{(2)}) \right), \\ \exists \left(\mathbf{s}_i^{(1)} = \mathbf{s}_j^{(2)} \right) \ \forall \, i, j \in \{1, 2\} \end{array} \right\}.$$

Figure 6.6 shows how nodes are connected following this method.

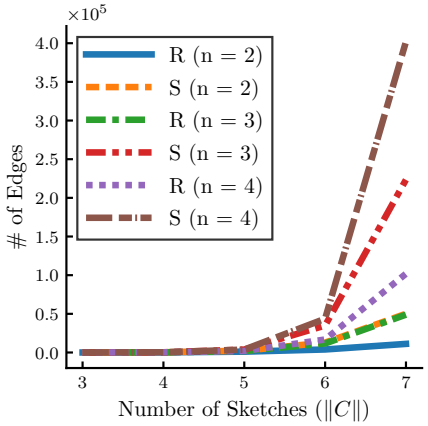
The simple edge creation method is a first approach at building tractable Bayesian Networks capable of inferring Sketch similarities. Through a correlation analysis of the resulting nodes and the similarities they represent, we develop a restrictive edge creation method that improves over our former simple edge creation method. The *restrictive edge creation method* adds edges between nodes following the simple edge creation method but adds a new condition. Edges are added if, additionally, it is true that for nodes $(\mathbf{s}_1^{(1)} | \mathbf{s}_2^{(1)}), (\mathbf{s}_1^{(2)} | \mathbf{s}_2^{(2)})$ they meet either $\mathbf{s}_i^{(1)} = \mathbf{s}_i^{(2)}$ and $\mathbf{s}_j^{(1)} \subset \mathbf{s}_j^{(2)}$ or $\mathbf{s}_i^{(1)} = \mathbf{s}_j^{(2)}$ and $\mathbf{s}_j^{(1)} \subset \mathbf{s}_i^{(2)}$. In the small scenario of Figure 6.6, the restrictive edge creation method connects the nodes with the same edges as the simple method.

restrictive edge creation method

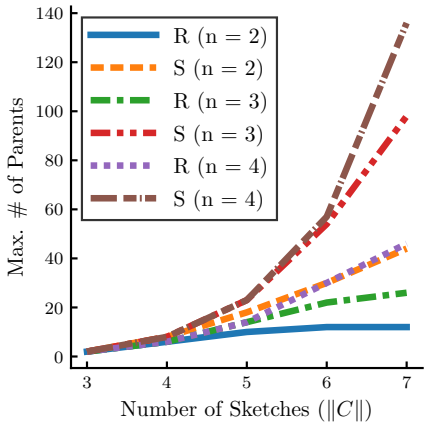
The benefits of using the restrictive edge creation method are most noticeable when multiple Sketches and many aggregations are taken into account. This fact is illustrated in Figure 6.7 by showing the performance of the two edge creation methods. In both methods, edges are added between nodes that are created using different combinations of the parameters n and $\mathbf{C}$ of our proposed node creation process. In the figure, we denote the simple method with an S and the restrictive method with an R. In parenthesis, we display the maximum number of aggregations (the n parameter) used in the node creation process. All x-axes show the number of Sketches (the $\|\mathbf{C}\|$ parameter) used in the node creation process.

Each plot in Figure 6.7 shows that the proposed restrictive edge creation method (R) achieves better overall results. In Figure 6.7a, we show the total number of edges created. In this context, the restrictive method performs exponentially better than the simple method. In Figure 6.7b and 6.7c, we show the maximum number of parents of any node and the average number of parents of all nodes, respectively. With any combination of parameters, the restrictive edge creation method yields a network with less parents and, thereafter, more tractable.

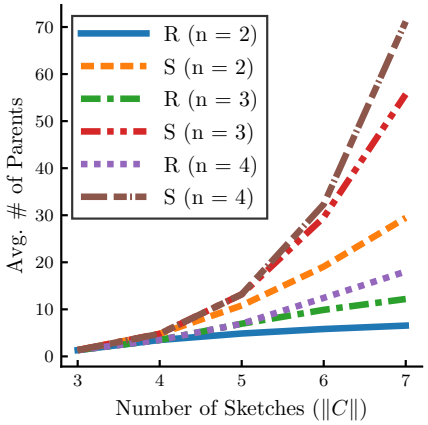
Tractability is far from being the most important metric for comparing Bayesian Network models. The likelihood of the Bayesian Networks, given a dataset for them to model, is a widespread metric used for comparisons [Koller et al., 2009]. The likelihood is normally penalized by the total number of model parameters to avoid overfitting (e. g., MDL, AIC and BIC [Z. Liu et al., 2012]). Having less of, both, total and average parents implies that a Bayesian Network has smaller CPTs and, therefore, less parameters. In all our experiments, both simple and restrictive edge creation methods achieved the same likelihood scores for the same datasets. As a consequence, we claim



(a) Total number of Edges created.



(b) Max. number of parents a node has.



(c) Avg. number of parents per node.

Figure 6.7: Performance of the Restrictive (R) and Simple (S) edge creation methods. In between parenthesis, we show the maximum aggregations n a Sketch may have. Overall, the Restrictive method performs better than the Simple method.

that the restrictive method correctly prunes useless edges as it has less parameters than the other method.

6.6.1.3 *Node Structure and Parameter Sharing*

Large Bayesian Networks are difficult to train and utilize. The larger the network, the more data is needed to accurately estimate the network parameters θ. Additionally, due to the (NP-hard) complexity of most inference mechanisms [Reed et al., 2014], large networks are restricted to the application of approximate, often inaccurate, inference techniques. However, the negative effects of large networks are mitigated the more parameters the nodes of a Bayesian Network share [Koller et al., 2009]. The previous node creation (Section 6.6.1.1) and connection (Section 6.6.1.2) mechanisms present the opportunity of creating nodes that can share their parameters.

While learning the parameters of Bayesian Networks (Section 6.6.2), we observed that when nodes share a specific set of properties they also share the same parameters. We define this set of properties as the *node stereotype*. The node stereotype is composed by the *rank* and *number of parents* of a node. For a given combination of parameters

node stereotype

$\|\mathbf{C}\|$ and n maximum Sketch aggregations, of the node creation mechanism, we observe few stereotypes (in contrast to the total number of nodes). Figure 6.8 shown the number of node stereotypes a Bayesian network has in relation to the maximum Sketch aggregation n. The number of node stereotypes remains constant no matter the number of Sketches $\|\mathbf{C}\|$. The number of stereotypes grows quadratically according to the maximum Sketch aggregations n. Given the amount of shared parameters, Bayesian Networks become tractable for as large as the set of Sketches $\mathbf{C}$ may be and for values of n that generate thousands of nodes. Example 6.3 shows an example of a Bayesian Network that becomes tractable due to parameter sharing. In Example 6.4, we show how nodes with the same stereotype look like.

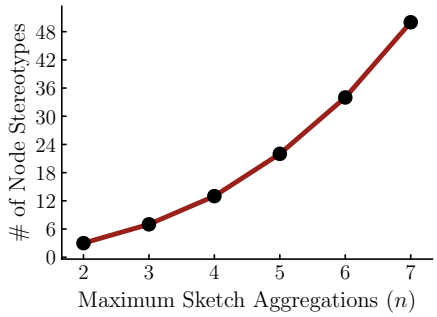


Figure 6.8: The number of node stereotypes found in a Bayesian Network that is constructed using a maximum of n Sketch aggregations.

Example 6.3: A Bayesian Network Made Tractable

Assume that the nodes of a Bayesian Network are generated as previously explained in Section 6.6.1.1 and connected with the *restrictive* edge creation methodology (Section 6.6.1.2). With six Sketches ($\|\mathbf{C}\| = 6$) and a maximum of four Sketch aggregations $n = 4$, we find 16 node stereotypes. A total of 1,500 nodes and 5,000 edges are created. Despite the large number of nodes and edges, because of having only 16 stereotypes, we only need to calculate the parameters of 16 random variables shared among all nodes.

Example 6.4: How Node Stereotypes Look

The Bayesian Network shown in Figure 6.6 has three stereotypes (indicated by the shading of the node). The first layer is composed of nodes of rank $(1, 1)$ and zero parents. The second layer has nodes of rank $(1, 2)$, each with two parents. The last layer only contains nodes of rank $(2, 2)$ as well as two parents. As such, according to the knowledge we developed about node stereotypes, the Bayesian Network is composed of three random variables (instead of nine if we count one random variable for each node). From this simple example, we can understand the nature of node stereotypes: Nodes representing the same type of Sketch compar-

ison are to be considered as having the same probability distribution (parameters).

6.6.2 *Learning the Bayesian Network Parameters*

The nodes of the Bayesian Network are random variables that represent the similarity between two Sketches (aggregated or not). We choose to model the similarities of Sketches with discrete random variables instead of continuous ones. Discrete random variables are easier to learn and do not require that we make assumptions about their PDF. As a consequence, the parameter θ of the Bayesian Network is composed of discrete CPTs. We define the *similarity* $\psi : \phi \to \mathbb{N}^+$ between two Sketches as $\psi\left(\phi\left(\mathcal{S}^{(X)}, \mathcal{S}^{(Y)}\right)\right)$ or, using a shorthand notation, as $\psi(X, Y)$. The similarity ψ is a discrete mapping from the Sketch divergence ϕ to a positive integer. As the mapping function ψ, we choose a step function with equidistant steps. Example 6.5 shows a three-step equidistant function. Example 6.6 shows how CPTs would look like when Equation 6.3 is used.

Sketch similarity ψ

Example 6.5: A Step Function with Three Equidistant Steps

A distance function ψ with three equidistant steps would look like Equation 6.3.

$$\psi\left(\phi\left(\mathcal{S}^{(X)}, \mathcal{S}^{(Y)}\right)\right) = \begin{cases} 1 & \text{if } 0.00 \leqslant \phi\left(\mathcal{S}^{(X)}, \mathcal{S}^{(Y)}\right) \leqslant 0.33 \\ 2 & \text{if } 0.33 < \phi\left(\mathcal{S}^{(X)}, \mathcal{S}^{(Y)}\right) \leqslant 0.66 \\ 3 & \text{if } 0.66 < \phi\left(\mathcal{S}^{(X)}, \mathcal{S}^{(Y)}\right) \leqslant 1.00. \end{cases} \tag{6.3}$$

6.6.2.1 *Learning from Assumptions*

In the context of a Bayesian Network, learning is the process of calculating the parameters of nodes using Maximum Likelihood Estimation (MLE) on top of samples of the random variables represented by the nodes. If we are able to assume the distribution of the Sketch similarities, we can create a dataset with which to calculate the parameters of the nodes using MLE. The process of learning from assumptions is as follows. First, we create a dataset of Sketch similarities from similarity assumptions. Second, we use MLE to compute θ. These two steps yield CPTs that express the probability distributions of Sketch similarities.

To learn the parameters of a Bayesian Network from assumptions, we create a dataset of Sketch similarity samples (that follow some assumed distribution family). The dataset creation algorithm is shown in Algorithm 6.1. The algorithm expects three input parameters: a set of empty Sketches $\mathbf{C}$, the maximum aggregation of Sketches n, and

Example 6.6: The Parameters of Discrete CPTs

The CPTs of nodes $(A|C)$ and $(A|BC)$ of Figure 6.6, when three discretization steps are used, may look like those in Table 6.3. With no parents, the CPT of node $(A|C)$ has three parameters. With two parents, the CPT of node $(A|BC)$ has 27 parameters.

$P(\psi_1 = x_1)$	
$x_1 = 1$	0.52
$x_1 = 2$	0.23
$x_1 = 3$	0.25

(a) CPT of the node $(A|C)$ where $\psi_1 = \psi(A, C)$.

$P(\psi_3 = x_3 \mid \psi_2 = x_2, \psi_1 = x_1)$	
$x_3 = 1, x_2 = 1, x_1 = 1$	0.03
$x_3 = 1, x_2 = 1, x_1 = 2$	0.03
$x_3 = 1, x_2 = 1, x_1 = 3$	0.02
...	...
$x_3 = 3, x_2 = 3, x_1 = 3$	0.04

(b) CPT of the node $(A|BC)$ where $\psi_1 = \psi(A, C)$, $\psi_2 = \psi(A, B)$ and $\psi_3 = \psi(A, BC)$.

Table 6.3: Example CPTs of two nodes of the Bayesian Network when nodes have up to three distinct similarities.

```
input  : Set of empty Sketches C
input  : Maximum number of Sketch aggregations n
input  : Desired number of dataset samples d
output : Dataset of examples of Sketch similarities D
 1  D ← ∅;
 2  X ← GenerateNodes(C, n);
 3  for i to d do
 4      for each Sketch c ∈ C do
 5          c ← EmptySketch();
 6          EncodeFeatures(c, SampleFeature())
 7      sample ← ∅;
 8      for each node x ∈ X do
 9          sample ← sample ∪ NodeSimilarity(x, C);
10      D ← D ∪ { sample };
11  return D
```

Algorithm 6.1: Procedure for creating a dataset of Sketch similarity samples useful for learning the parameters of a Bayesian Network that represents the probabilistic relationships between Sketch similarities.

the number of samples to generate d. In line 2, we generate the nodes of a Bayesian Network as described in Section 6.6.1.1. For each desired dataset sample d, in line 6, `EncodeFeatures()` encodes features sampled from assumed distribution families. The logic of the feature

sampling process is captured within `SampleFeature()`[3]. As a result, each sample contains the similarity comparisons made by each node of the Bayesian Network. The combination of all these samples makes up the training dataset. This training dataset is used to learn the parameters θ using an MLE variant that copes with shared parameters [Koller et al., 2009].

> **Example 6.7: How Does a Sample Look Like**
>
> Given node $(X|YZ)$, we aggregate Sketches $Y, Z \in \mathbf{C}$ (as explained in Section 6.6.1.1) and proceed to calculate the similarity of the aggregation against Sketch $X \in \mathbf{C}$. The similarity is calculated by a function such as Equation 6.3 using as many steps as desired.

6.6.2.2 *Learning from Real Data*

Learning the parameters θ from real data is not substantially different than learning from assumptions. The only difference lies in how `SampleFeature()` forwards the samples to `EncodeFeatures()` in line 6 of Algorithm 6.1. When real data is available, the data itself is used to extract features and populate every Sketch $c \in \mathbf{C}$. In the evaluation section of this chapter (Section 6.8), we demonstrate how hosts experiencing similar communication patterns can be found in a large backbone network.

6.7 INFORMATION DISSEMINATION: PROBABILISTIC FORWARDING OF SKETCHES

The last sections describe two of the three components that comprise our system for disseminating similarities. This sections describes the last component and how the other two are jointly used to disseminate similarities. To better understand the dissemination strategy, we distinguish between two types of Sketches. *Local Sketches* are those that encode the feature counts of local observations. From the perspective of an individual member, a *Remote Sketch* is any Sketch created by a remote member that, because of the dissemination process, is now in the possession of the member.

local Sketch

remote Sketch

Local Sketches are used for two purposes. First, they are used to compute similarities between them and remote Sketches. Second, they are aggregated with remote Sketches and disseminated to other members. With remote Sketches, members can deduce the similarity between their local observations and the aggregated observations of others. By using the similarity deduction component (Section 6.6), we can deduce the similarity between the local observations and an individual member that participated in the aggregation of a remote

3 For example, features could be assumed to follow a family of Gaussian distributions where each individual distribution may have different parameters.

probabilistic Sketch forwarding

Sketch. To tie everything together, we propose an algorithm called *Probabilistic Sketch Forwarding*. This algorithm determines how aggregated Sketches are forwarded to other members.

```
input : Number of forwarding rounds r
input : Number of maximum chain length c
input : Sketch forwarding probability p
 1 for 1 to r do
 2     m ← RandomMember(∅);
 3     Sk ← GetSketch(m);
 4     Kn ← GetKnowledge(m);
 5     chain ← { m };
 6     do
 7         m ← RandomMember(chain);
 8         m.UpdateKnowledge(chain, Sk, Kn);
 9         Sk ← Aggregate(Sk, GetSketch(m));
10         Kn ← Combine(Kn, GetKnowledge(m));
11         chain ← chain ∪ m;
12     while ‖ chain ‖ < c and Random(0, 1) < p;
13 return
```

Algorithm 6.2: Probabilistic Sketch forwarding algorithm.

In Algorithm 6.2, we show the proposed Sketch forwarding algorithm. The algorithm is shown as if operated by a central oracle (for the sake of simplicity). This does not detract from applying the illustrated concepts in distributed environments. The algorithm is based on the concept of building chains of Sketches throughout different rounds. Sketches are chained together through the process of aggregation. In each round of the algorithm, a new chain of Sketches is formed as Sketches are received, aggregated and forwarded.

The operation of the algorithm depends on three user-supplied parameters. The number of rounds r specifies how many Sketch chains are formed. The maximum chain length c indicates how long Sketch chains can be. The parameter c is related to the maximum number of Sketch aggregations n, as introduced in Section 6.6.1.1, such that $c = n + 1$. Finally, the forwarding probability p specifies the probability of expanding a chain or not. In each round, a Sketch chain is initiated by randomly selecting an initial member and using its local Sketch as the base Sketch of a new chain (lines 2 to 5). Additionally, some knowledge about this initial member is obtained. Knowledge refers to the Sketch similarities $\psi(X, Y)$ of Sketches X and Y that are already known (because they were directly observed or deduced with high certainty). Sketch chains are forwarded to a random member that has not already taken part in the current chain (line 7). The chain receiver updates its knowledge using what is known about the members participating in the chain, and the aggregated Sketch itself (line 8). Notice that only three data structures need to be sent

over the network to other members for each new link of a chain: a list of members participating in the chain, the aggregated Sketch of the participating members, and the combined knowledge of all chain members. Finally, in lines 9 and 10, a new member is added to the chain. With probability p, the chain is forwarded to another member (line 12).

6.8 EVALUATION

In this section, we put together the three components that constitute our system and evaluate their performance. We demonstrate the capabilities of the system to distributedly identify collaborating members that observe similar features using less information than flooding mechanisms. Our evaluations demonstrate this when synthetic and real-world data are used.

6.8.1 *Experimental Setup*

Every experiment we conduct has four preparatory steps. In the first step, using using Algorithm 6.1, we generate a dataset of feature observations either from assumptions or real-world data. In the second step, we encode the features of each collaborating member $c \in \mathbf{C}$ into a Sketch as detailed in Section 6.5. In the third step, we build a Bayesian Network following the methods described in Section 6.6. The nodes of the Bayesian Network are created for a given number of members $\|\mathbf{C}\|$ and maximum Sketch aggregations n. The nodes are connected in accordance to the edge building methodology in Section 6.6.1.2. We learn the parameters of the Bayesian Network using the dataset created in the first step. In the fourth and last step, collaborating members send, collect and aggregate Sketches (for later forwarding) to disseminate knowledge. We follow the algorithm presented in Section 6.7 to simulate how members forward Sketches in a network.

In all experiments, when learning the parameters of the Bayesian Network architecture we propose, we discretize the divergence ϕ between two Sketches so that their similarity ψ has five possible values (cf. Equation 6.3). Flooding is used as a baseline to measure the efficiency of our methodology with respect to communication overhead. Given a set of Sketches such as $\mathbf{C} = \{A, B, C\}$, we calculate that to achieve full knowledge using flooding, a total of $f(\mathbf{C}) = \|\mathbf{C}\| \times (\|\mathbf{C}\| - 1) = 6$ messages need to be sent. We compare the accuracy of our deductions taking into account how many messages are exchanged relative to $f(\mathbf{C})$. If our deduction strategy uses four messages instead of the six needed by flooding, we say that 66.66 percent of messages, with respect to f, are used. The accuracy of the system is measured by calculating the average prediction error of the similar-

flooding as a baseline

accuracy metric

ities. More specifically, after all members have received all Sketches chosen to be sent over the network, each member predicts the similarity of itself against all other members. The accuracy is the average error of all predictions. In all experiments, the standard deviations of the average errors are displayed. Furthermore, the average errors are compared against the baseline of the accuracy that would be achieved if similarities are chosen randomly.

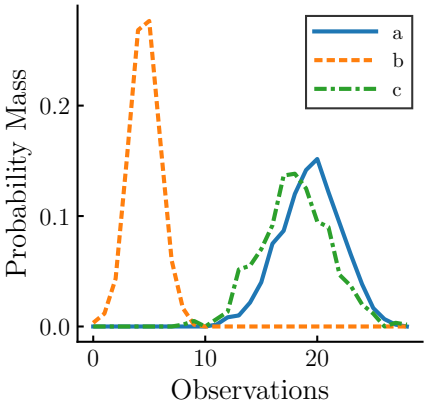
random baseline

6.8.2 *Deductions using Assumptions*

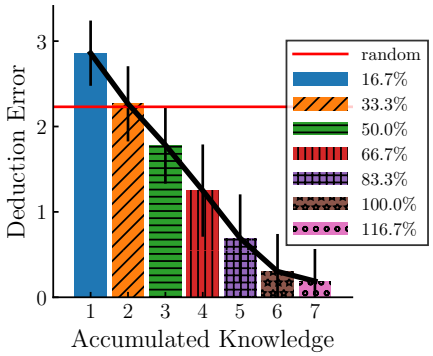
When learning the parameters of a Bayesian Network using assumptions, the Bayesian Network can accurately deduce similarities using less messages than what flooding would require. To demonstrate this claim, we test a setup of three collaborating members $\mathbf{C} = \{A, B, C\}$ aggregating no more than two Sketches $n = 2$. We begin by generating a dataset of $500,000$ samples (using Algorithm 6.1) with the underlying assumption that features are distributed according to a family of Gaussians. We then build a Bayesian Network following the methodology presented in Section 6.6 with the algorithm parameters $\|\mathbf{C}\| = 3$ (three members) and $n = 2$ (up to two Sketch aggregations). The previously generated dataset is used to learn the CPT parameters θ of the Bayesian Network using MLE.

We test two different scenarios involving three collaborating members. The scenarios and their evaluations are shown in Figure 6.9. In *Scenario 1* (top row), members A and C share features with each other but not with B. In *Scenario 2* (bottom row), the features of C have few similarities between the features of A and B. None of the features of A and B are shared. In Figure 6.9a and 6.9c, we show the PMF of the features of each member. The accuracy of the deductions made by our strategy are shown in Figure 6.9b and 6.9d. Each bar in the plot shows the average deduction error (y-axis) when a specific number of messages are exchanged (x-axis). Each bar is labeled after the percentage of messages exchanged in contrast to flooding. For example, at 100 percent, the same number of messages are sent as with flooding. The horizontal line shows the average deduction error if we choose random similarities.

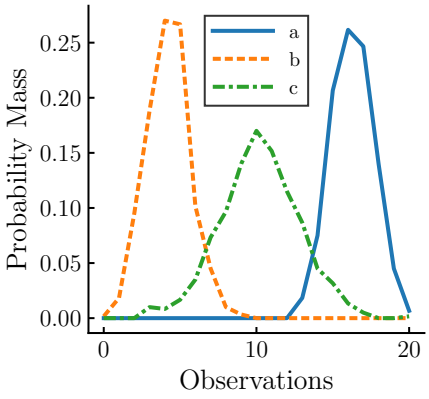
From both Figure 6.9b and 6.9d, we can appreciate that when sending 66 percent of the messages needed by flooding, the average error (including its standard deviation) is below the *random baseline* (red horizontal line). With 83 percent of the sent messages, the average error has the tendency of being below one. This implies that the deduction mechanism, using close to 20 percent less information, can deduce all the similarities between the members with only few exceptions. The exceptions, nonetheless, are never far away from the true value.



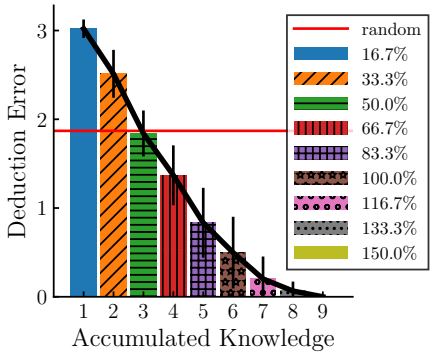
(a) Scenario 1: Members a and c record similar observations. Member b does not share any observations with the others.



(b) Deduction error against the number of shared messages (accumulated knowledge) of observations in (a).



(c) Scenario 2: Member *c* records subtle, yet different, observation similarities with a and b.



(d) Deduction error against the number of shared messages (accumulated knowledge) of observations in (c).

Figure 6.9: Deduction error using data assumptions with different types of observation overlaps for three collaborating members $\|\mathbf{C}\| = 3$ and a maximum Sketch aggregation of two $n = 2$.

When increasing the number of members $\|\mathbf{C}\|$ and maximum number of Sketch aggregations n, we observe the same trend: With 20 percent less messages than what flooding sends to achieve full knowledge, our strategy estimates the similarity of each member accurately with only small errors. Moreover, from the standard deviation, we can recognize that errors are small. Figure 6.10 shows the deduction error when using four members when averaged over multiple random scenarios. Each scenario corresponds to a different configuration of the features shared by the members. It is worth noting that, when considering more members, the accuracy of the system is still below

the *random baseline* (red horizontal line) even when a small set of messages are exchanged.

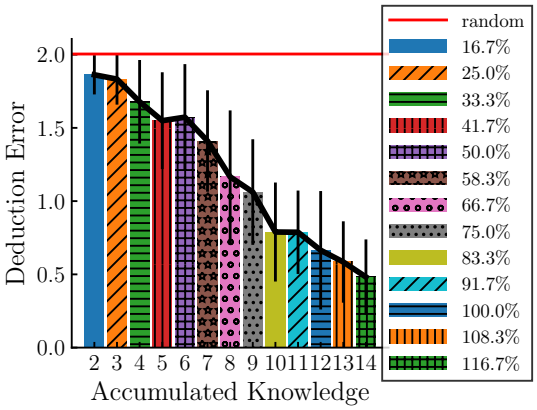


Figure 6.10: Deduction error of the average of multiple scenarios in relation to how feature observations overlap. We consider four collaborating members $\|\mathbf{C}\| = 4$ and a maximum of two Sketch aggregations $n = 2$.

This and the previous experiments confirm that our dissemination strategy is able to start estimating the similarity of collaborating members as soon as information arrives, without having to wait for the full information. This is evident from how the deduction error decreases with each new received message. Furthermore, less messages are required than what flooding needs to achieve an approximation to the real similarities. It is however important to mention that our strategy is stochastic in how Sketches are encoded and in the probabilistic inference of the Bayesian Network. This implies that even if 100 percent of information is received, in contrast to the flooding mechanism, errors can still be made; albeit small.

6.8.3 *Deductions using Real-world Data*

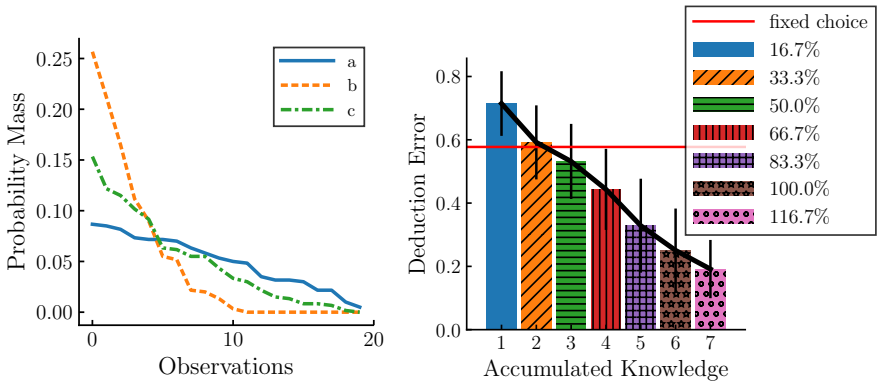
Real-world observations can be used to learn the parameters of the Bayesian Networks we propose to deduce similarities (Section 6.6). Our Bayesian Networks are as accurate in deducing similarities when their parameters are learned from real-world observations instead of assumptions. To demonstrate this claim, we test the capability of our methodology of finding similar traffic patterns in an Internet backbone network. We begin by sampling different days of a publicly available network backbone dataset known as the MAWI archives [Cho et al., 2000]. With the samples, we create a new dataset for learning the parameters of our proposed Bayesian Network. Afterwards, using the Bayesian Network, we deduce the similarity of the traffic patterns observed in three different days not used while creating our dataset. We measure the accuracy of the deductions when different amounts of information are shared.

A Bayesian Network is designed to deduce the similarity of three collaborating members $\|\mathbf{C}\| = 3$ and up to two Sketch aggregations $n = 2$ following the methodology of Section 6.6.1. The parameters

of the network are learned using 15 days from September 2017 of the MAWI archives. From each day, in windows of one minute, we record the counts of all destination IP addresses observed. The counts are sorted from high to low and only the 20 most seen IPs are kept for each day. Given that each day of the MAWI archives has 15 minutes of anonymized traffic, a total of 45 counts are collected.

From the 45 total IP counts collected, we derive a dataset to learn the parameters of our Bayesian Network. We follow Algorithm 6.1 for this purpose and detail each step referencing lines of the algorithm. To create each sample of our dataset, we pick three different IP counts and encode each one into a Sketch (line 6). Then, for each node of the designed Bayesian Network (line 8), the Sketches are used to calculate the similarity of the node (line 9). All these similarities are recorded as one sample and stored as part of the dataset (line 10). Because ${}^{45}C_3 = 14,190$, our dataset consists of 14,190 samples. MLE is used on top of these samples to learn the shared parameters θ of the Bayesian Network [Koller et al., 2009].

In contrast to the setup used for evaluating the accuracy of our deductions using data assumptions, we modify the similarity function ψ (Section 6.6.2). From empirical studies on the MAWI archives, we observed that the distribution of destination IP addresses tend to overlap. Because of this, we choose to discretize the divergence of two Sketches $\phi(X, Y) \rightarrow [0, 1]$ more tightly using the exponentially spaced bins $[0.00, 0.09, 0.24, 0.47, 0.84, 1.00]$. This way, for example, for two Sketches to have a similarity of one, their divergence would need to be in between 0.00 and 0.09.



(a) Real-data Scenario: Members observe different destination IP distributions.

(b) Deduction error compared to the number of shared messages (x-axis) of observations in (a).

Figure 6.11: Deduction accuracy when using real-world data to test the deduction capabilities of three collaborating members $\|\mathbf{C}\| = 3$, considering up to two Sketch aggregations $n = 2$.

Having learned the parameters of the Bayesian Network, we can deduce the similarity of three arbitrary IP counts (not observed during

the dataset generation process). In Figure 6.11, we evaluate our deduction mechanism using three one minute IP counts of destination IP addresses taken from three days in October 2017 from the MAWI archives. The PMFs of the 20 destination IP addresses with the most observations are shown in Figure 6.11a. We simulate each count belonging to a different collaborating member (among the three). We employ our Probabilistic Forwarding algorithm (Algorithm 6.2) to determine how messages are disseminated among the collaborating members. In total, we simulate 250 different message dissemination configurations. We show in Figure 6.11b the average deduction error (and its corresponding standard deviation) using all message passing configurations. As a baseline, we compare the accuracy of our methodology against the accuracy obtained if we always predict that the similarity of two counts is two ($\psi(X, Y) = 2$). This fixed choice yields the best average accuracy among all other possible fixed choices. The results indicate that we can predict the similarity of all counts using 20 percent less information than flooding.

Our approach can deduce which members have observed similar feature counts even in large backbone networks. Finding similar traffic patterns opens the possibility of detecting many intrusions that would typically be difficult to detect. For example, if we follow the same approach as the one used in SkyShield [Cho et al., 2000], it would be possible to collaboratively identify DDoS attacks occurring in different parts of a network without the need of a centralized entity. That is, if we identify that one collaborating member is experiencing a DDoS attack, by deducing the similarities of the traffic characteristics between the affected member and all others, DDoS attacks targeting other members can also be spotted.

6.9 CONCLUSION AND LESSONS LEARNED

Early on while reviewing related work on CIDSs, we identified the need of a similarity identification strategy that would send less messages than what flooding needed and that did not rely on centralized components. Most researchers do not detail how information is disseminated in the process of determining the similarities of members within a CIDS; instead, researchers take this for granted and build a system on top of this non-existent component. In this chapter, we focused on creating such a similarity identification strategy. In the process of designing the strategy, however, we encountered multiple problems with non-trivial solutions. In the end, what is presented herein is the result of solving different problems through three tasks. First, we searched for a common denominator that CIDSs use to identify similarities. Second, we analysed multiple correlation and aggregation techniques and selected one. Lastly, we studied diverse ways to deduce information from correlations and aggregations.

three tasks

first task

As the first task, through an analysis of related work, we identified that *member similarities* were the common denominator that many CIDSs use to detect intrusions. Intrusion detection, at its core, is achieved by allowing members to share information among themselves and letting the members perform aggregation and correlation on the information. However, we were not able to find non-trivial solutions[4] when considering the general problem of sharing raw information. Instead, we focused on sharing feature counts (or histograms) and enabling members to aggregate and correlate these. This decision was made after we observed that most CIDSs only require feature counts for performing similarity comparisons.

second task

Having in mind that we only wanted to share feature counts, we determined as the second task that we needed a data structure specialized on summarizing counts. Furthermore, the data structure needed two specific properties. One, it needed to combine with others of its type to create aggregates. Second, it could calculate how different it was from others, aggregated or not. Bloom filters were immediately discarded as they were not able to track counts. Their counting variants were also not suitable as they lacked a well-defined comparison operator. The count-min Sketch PDS was chosen as it had these two properties.

By only sharing aggregates, we could lower the communication overhead significantly. However, aggregates would loose enough information that after three or more aggregated feature counts (in a single Sketch), the accuracy of calculating similarities dropped significantly. From this observation, we recognized the need of a deduction mechanism able to determine how members contributed to an aggregate. We studied the concept of members contributing to an aggregate, where we determined that it was possible to manually detect who was responsible for increasing or decreasing the similarity of an aggregate. As the third task, we searched for a mechanism capable of doing inference on Sketch aggregations. We experimented with different ML techniques such as convolutional neural networks, deep autoencoders and probabilistic graphical models. Our experiments with Bayesian Networks (a type of probabilistic graphical model) yielded the most promising results.

third task

Designing the Bayesian Networks needed to deduce Sketch similarities was a difficult task on its own. When designing a Bayesian Network, two main challenges need to be overcome. First, we need to determine the random variables that nodes will represent. Second, nodes need to be connected so that they form a Directed Acyclic Graph (DAG). The decision of how to model nodes impacts how they are connected, and vice versa. From a mathematical perspective, we concluded that continuous random variables would not scale to large

4 A trivial solution to reduce the overhead of sharing information would involve compressing all information before it is shared.

problems spaces. We settled for the discretization of similarities as random variables.

After choosing how to represent the nodes of our Bayesian Networks, we experimented with different ways of connecting the nodes to form DAGs. We quickly found a mechanism that would connect all nodes, using as few edges as possible, maximizing the likelihood of the probabilities they represent. However, the resulting networks would be large and intractable for more than a few nodes. After manually analysing the resulting networks, we discovered that the CPTs represented by some nodes were the same in several circumstances. After further analysis, we finally discovered that nodes with the same number of parents and *node rank* (see Section 6.6.1.1) had shared parameters. We established the term *stereotype of a node* to distinguish sets of nodes that share these two properties. This was the final piece that enabled tractable Bayesian Networks.

6.9.1 *Future Work*

Our dissemination strategy is underlay agnostic. The probabilistic forwarding algorithm could be designed to take into account the network underlay and minimize the number of hops messages take in a network. The forwarding algorithm would also benefit from a stochastic technique that can make sure that, on average, each member receives an equal amount of messages.

6.9.2 *Chapter Summary*

The description of our system began in Section 6.5 with a demonstration of how feature counts are locally encoded using Sketches. Afterwards, in Section 6.6, we proposed a Bayesian Network design capable of deducing the similarities of Sketches that encode feature counts. With the final proposition of a mechanism to probabilistically aggregate and disseminate Sketches through a network, we arrived at a methodology capable of deducing the similarity of local observations when only partial observations are shared. This had the benefit of reducing the network footprint close to 20 percent when compared to the number of messages that would need to be sent to guarantee full knowledge. Furthermore, it was possible to start deducing similarities as soon as information is made available without having to wait for the dissemination of information to converge. The proposed strategy enables a viable communication strategy between collaborating members of a CIDSs that respects all CIDS requirements plus all additional requirements we have proposed.

7 COLLUSION DETECTION

CONTEXT

In the last chapter, we developed a dissemination mechanism tailored to the distribution of information between members of a CIDS. Disseminating information is indeed the key to collaboration between CIDS members. The dissemination mechanism of the last chapter, however, makes the implicit assumption that CIDS members are always honest. Honest members pursue the common goal of detecting collaborative attacks without hidden agendas. In practical scenarios, we need to acknowledged the existence of hidden agendas and discourage dishonest members from acting against the common goal. Furthermore, to complicate matters, we need to consider the scenario where dishonest members collude to push their personal or collective agendas.

Honesty is a virtue that we cannot assume is present in all the members of a CIDS. In this chapter, we acknowledge that acting dishonestly within a CIDSs is a possibility and develop a trust mechanism to detect single dishonest members as well as groups of colluding members. In our trust mechanism, collaborating members rate the reliability of each other to determine trust scores. A member with a low trust score signals that the member chooses to be unreliable to hide information or that it is giving inaccurate ratings on purpose.

Figure 7.1 shows an overview of this chapter in relation to the thesis as a whole. This chapter is the fifth contribution of this thesis. The contribution relates to the *Membership Management* layer of the CIDS model we reference (see Section 2.3.3). In this chapter, we develop a trust mechanism that uses the reliability of CIDS members to determine trust scores. CIDSs can use trust scores to manage the members of a CIDS. When trust scores go below a certain threshold, members can be blacklisted and prevented from participating in the exchange of information. We only focus on developing the trust mechanism and disregard the engineering task of using trust scores to create blacklists.

The structure of this chapter is organized as follows. We begin with a general introduction to the problem of building trust relationships within CIDSs. In Section 7.2 and Section 7.3, we present background relevant to this chapter; and related work in the field of bayesian trust models, ML for trust and trust management within CIDSs, respectively. Our trust mechanism for detecting colluders is explained

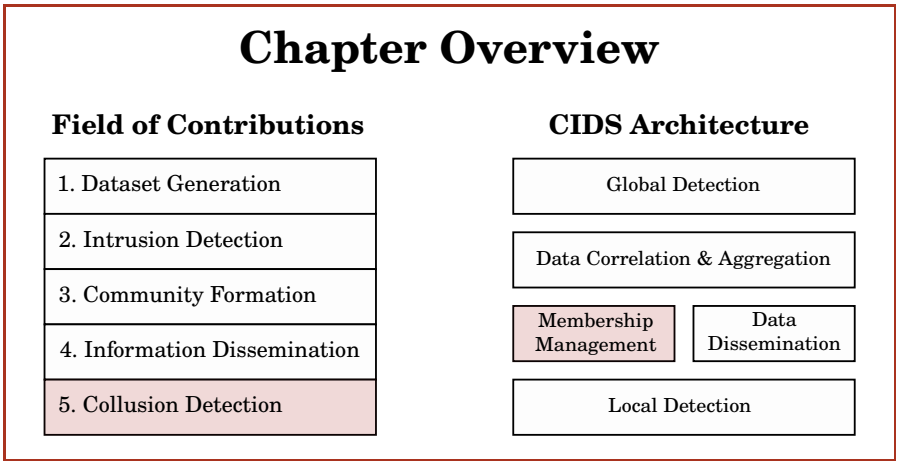


Figure 7.1: This chapter comprises the fifth contribution of this thesis: *Collustion Detection*. The contribution is tied to the highlighted layer of our referenced CIDS architecture: *Membership Management*.

in Section 7.4. A detailed evaluation of our mechanism follows in Section 7.5. We finish with a conclusion in Section 7.6 which containing our lessons learned and limitations.

7.1 INTRODUCTION

Many recent massive cyber-attacks have highlighted the need of large-scale proactive security. The Mirai botnet, for example, affected many Internet sites and services in 2016 with massive DDoS attacks of up to 600 Gbps in size. CIDSs can detect and, with the knowledge they generate, enable third parties to mitigate the effects of such attacks. At their core, CIDSs enable multiple independent sensors (i. e., NIDSs, honeypots or firewalls) to share information to create holistic views of large networks. From these holistic views, distributed and more sophisticated attacks can be identified and prevented. Sharing information is therefore the key component that enables CIDSs.

honest sensors

dishonest sensors

coalition

Sharing information is, however, only effective if all collaborating sensors in a CIDS are honest. *Honest sensors* follow the common goal, or agenda, of sharing information as accurate as possible in order to detect distributed attacks. Conversely, *dishonest sensors* may choose to share information that would advance their personal agendas. Dishonest sensors may further alter the information they share or collude with other dishonest sensors. A group of dishonest sensors that colludes to advance an agenda that contradicts the agenda of the honest sensors is known as a *coalition*.

7.1.1 *Problem Statement*

Dealing with the sharing of unreliable or false information by dishonest sensors is a major challenge in the field of CIDSs. The accuracy of CIDSs heavily relies on the quality and accuracy of shared information. Inaccurate or misleading information can severely degrade the overall detection capabilities of the system [Vasilomanolakis, Karuppayah, et al., 2015]. CIDSs need a membership management mechanism to remove sensors that degrade the performance of the system.

Many CIDSs use trust mechanisms to detect dishonest sensors, e. g., [C. J. Fung, J. Zhang, et al., 2011]. Trust mechanisms enable the sensors of a CIDS to rate each other with the goal of creating a holistic rating of each sensor. This holistic rating, which we term *trust score*, is meant to reflect what the CIDS believes the intentions of a sensor in the system are. In a suitable trust mechanism for CIDSs, a sensor with a high trust is a sensor that a CIDS considers honest, whereas a sensor with low score is assumed to be dishonest.

trust score

We use the reliability of sensors to compute trust scores. The *reliability* of a sensor is a property that quantifies the ability of a sensor to detect network attacks and is measured with the help of reliability traffic. *Reliability traffic* is network traffic that exhibits the properties (e. g., a signature) of known attacks which a sensor should be able to identify as malicious. We make the assumption that a CIDS sensor can insert reliability traffic into the network and that sensors cannot distinguish between normal and reliability traffic. A sensor keeps track of whether other sensors raise an alarm concerning the reliability traffic they inserted. Note that each sensor keeps track of the reliability of all other sensors. Therefore, sensors can compute a *rating* for each other sensor. The rating is a value in the range $[0, 1]$. A rating of zero means that a sensor is unreliable, i. e., the sensor reported no alarms even though it saw reliability traffic. An example of how sensors can create reliability traffic and how they can compute sensor ratings is shown in Example 7.1.

sensor reliability

reliability traffic

sensor rating

7.1.2 *Challenges*

A sensor may not issue an alarm in response to receiving reliability traffic due to several factors. Malice is certainly one factor, but incompetence, low sensor up-time, low network availability or limited computational resources are other possibilities. Reliability alone is therefore not enough to determine trust scores. We need a trust mechanism that takes into account reliability, without completely depending on it, to compute trust scores.

To further complicate matters, we consider the possibility of dishonest sensors forming coalitions. A coalition may choose to send negative reliability reports to artificially lower the trust scores of oth-

Example 7.1: Creating Reliability Traffic and Rating Sensors

Reliability traffic needs to contain attacks known to a sensor. A sensor can use ID2T, the result of the concepts presented in Chapter 3, to inject synthetic attacks into real traffic. ID2T blends synthetic attacks with real attacks such that others cannot recognize reliability traffic from normal one.

The process of creating reliability traffic could be as follows. First, a sensor randomly collects network traffic. Second, the sensor analyzes the traffic for intrusions. Third, if the traffic is free of intrusions, the sensor uses ID2T to inject it with one of 12 attacks (see Section 3.6). Finally, the sensor inserts the traffic with attacks back into the network and records which analyzers report alarms in response to the injected attack.

From the alarms issued (or not) by others in response to reliability traffic, a sensor can calculate the rating of a sensor. A simple technique to compute the rating is to set the rating as the percentage of correctly issued alarms. A more advanced technique involves a (Bayesian) probabilistic approach: An attack is associated with a prior $P(X)$ that relates to the likelihood of the difficulty of an attack $P(Y|X)$, where X and Y are the random variables, respectively, of *detecting an attack* and *difficulty of an attack*. The score of a sensor can then be the marginal probability computed from the Bayes theorem

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)}.$$

ers. A colluding group may also positively alter the reliability of their members to artificially increase their trust scores.

7.1.3 *Chapter Contributions*

Sphinx

We propose *Sphinx*[1], an evidence-based trust mechanism that uses the reliability of sensors within a CIDS to compute trust scores and, with these, uncover dishonest sensors. *Sphinx* does not only uncover individual dishonest sensors, but also coalitions of dishonest sensors. *Sphinx* is also oblivious to the underling CIDS architecture, being easily adapted to centralized or distributed systems. In contrast to the state of the art, e. g., [Duma et al., 2006], our system does not assume that honest or dishonest sensors are always consistent: a dishonest sensor might choose to act honestly with some probability.

1 In Greek mythology, a Sphinx was a creature that dwelt outside the city of Thebes, asking travelers a riddle to let them pass or be devoured. Our contribution is inspired in how the Sphinx asks riddles (questions with known answers) to make decisions.

7.2 SPECIALIZED BACKGROUND

In this section, we explain the preliminaries needed to understand our contribution. *Sphinx* relies on the two standard ML techniques of K-means clustering and GMMs to compute trust scores. The basics of these techniques are explained in what follows.

7.2.1 *K-means Clustering*

Let us define a dataset $\mathcal{P} = \{P_1, \ldots, P_n\}$ of n points in an Euclidean space of D dimensions, with $P_i = \{x_{1_i}, \ldots, x_{D_i}\}$, for $i = 1, \ldots, n$. With K-means clustering, we group the points in $\mathcal{P}$ into the K clusters $\mathcal{C}_1, \ldots, \mathcal{C}_K$, where each cluster minimizes the distance between the members of the cluster and its *center point*. More formally, for clusters $\mathcal{C}_1, \ldots, \mathcal{C}_K$, center points $M_1, \ldots, M_K$ are chosen, where $M_j = \{x_{1_j}, \ldots, x_{D_j}\}$, for $j = 1, \ldots, K$. Each center point M_j satisfies the property that the sum of the squares of the distances of each data point P_i to the closest point M_j is minimized. This concept can be formalized with the help of the *distortion measure* J, an objective function defined as:

$$J = \sum_{i=1}^{n} \sum_{j=1}^{K} r_{i,j} \left\| P_i - M_j \right\|^2,$$

where $\left\| P_i - M_j \right\|$ is the distance between the points P_i and M_j and where $r_{i,j} = 1$ if point P_i is assigned to cluster $\mathcal{C}_j$, otherwise $r_{i,l} = 0$.

The K-means clustering problem consists in finding values $r_{i,j}$ and centers M_j, for $i = 1, \ldots, n$ and $j = 1, \ldots, K$, such that the distortion measure J is minimized. This is achieved using the *EM* algorithm (see [Bishop, 2006]), which uses an expectation step *E* to adjust the values $r_{i,j}$, and a maximization step *M* to adjust the points M_j. The distortion measure J is iteratively minimized by further applying the *E* and *M* steps sequentially.

7.2.2 *Gaussian Mixture Models*

A GMM models the distribution of a dataset of points $\mathcal{P} = \{P_1, \ldots, P_n\}$, whose distribution would normally be difficult to model directly, using a linear combination of Gaussian distributions. Consider G Gaussian distributions $\mathcal{N}(\mu_1, \sigma_1^2), \ldots, \mathcal{N}(\mu_G, \sigma_G^2)$, with mean μ_i and variance σ_i^2, for $i = 1, \ldots, G$. A GMM fitted to dataset $\mathcal{P}$, denoted as $p(\mathcal{P})$, is defined as a linear combinations of the Gaussian distributions such that

$$p(\mathcal{P}) = \sum_{j=1}^{G} \pi_j \mathcal{N}(\mu_j, \sigma_j^2), \text{ where } \sum_{j=1}^{G} \pi_j = 1.$$

Each Gaussian $\mathcal{N}(\mu_j, \sigma_j^2)$ is known as a *component* of the mixture. Each component is associated with a *mixing coefficient* π_j. Because $\sum_{j=1}^{K} \pi_j = 1$, $p(\mathcal{P})$ is a probability distribution.

A Gaussian distribution $\mathcal{N}_{\mathcal{P}}(\mu, \sigma^2)$ approximates the GMM distribution $p(\mathcal{P})$ as closely as possible, according to the Kullback-Leibler divergence, when

$$\mu = \sum_{j=1}^{K} \pi_j \mu_j \quad \text{and} \quad \sigma^2 = \sum_{j=1}^{K} \pi_j (\sigma_j^2 + \mu_j^2) - \mu^2$$

(see [Lauritzen, 1996]). $\pi_1, \ldots, \pi_K$ correspond to the mixing coefficients of $p(\mathcal{P})$. *Sphinx* uses this fact in its computations of trust scores.

7.3 RELATED WORK

evidence-based trust mechanisms

When trust mechanisms rely on evidence derived from interactions, they are called *evidence-based trust mechanisms*. Evidence may be derived from either direct or indirect interaction. In a direct interaction, an entity collects evidence from personal experiences. In an indirect interaction, an entity collects evidence about the experiences a third party had with others. In this chapter, we are concerned with the collection of both direct and indirect evidence. *Sphinx* computes trust scores using direct and indirect observations of the reliability of sensors. In this section, we describe diverse evidence-based trust mechanisms based on statistical and ML techniques. We further highlight trust mechanisms applied within CIDSs.

7.3.1 *Bayesian Trust Models*

Bayesian trust models, i. e., [Hang et al., 2011; Nielsen et al., 2007; Ries, 2009], use Bayesian probabilities [Bolstad, 2004] to estimate the trust score of a trustee based on evidence from past interactions, e. g., [Josang et al., 2002]. These models, however, are not able to filter out dishonest evidence. Buchegg et al. [2004] proposes a reputation system that uses the honesty of the participants to build trust relationships. His reputation system uses the key idea of learning from how others act before considering direct interactions.

7.3.2 *Machine Learning for Trust Modeling*

ML has acquired an important role in the area of computational trust. Nowadays, an increasing amount of evidence (or data) is generated by large-scale web-based applications that rely on trust (e. g., Amazon, Ebay and Airbnb). To cope with the amount of generated data, researchers use ML to process enormous amounts of information and compute trust scores. However, even if evidence is not available, ML

can still be used to predict trust scores, e. g., [Xin Liu, Tredan, et al., 2014]. More specifically, ML can effectively solve the difficult problem of estimating the dynamic behavior of an entity from its interactions [Tang et al., 2012].

Sphinx uses ML techniques to calculate trust scores in the presence of honest and dishonest participants. With clustering algorithms, *Sphinx* identifies evidence submitted by dishonest participants. In contrast to the related work, with the technique of fitting a GMM to clusters, we identify unreliable evidence submitted by colluders. This confers *Sphinx* the capability of degrading the trust scores of dishonest sensors.

7.3.3 *Trust Management within CIDSs*

Early CIDS research had the assumption that collaborating members were reliable and honest by definition [Vasilomanolakis, Karuppayah, et al., 2015]. In recent years, the assumption of having no insider threats has been relaxed and the issue has been taken more seriously, e. g., [C. J. Fung, J. Zhang, et al., 2011; Gil Pérez et al., 2013]. However, related work has not extensively explored trust mechanisms based on reliability measurements. *Sphinx*, our contribution, is an evidence-based trust mechanism that focuses on managing trust using the reliability of sensors.

Fung et al. have worked on the identification of insider threats within CIDSs. In [C. J. Fung, Baysal, et al., 2008], they propose a framework to bestow CIDS sensors the ability to determine the trust score of others from past interactions. In [C. J. Fung, J. Zhang, et al., 2009], they propose an evidence-based trust mechanism that uses the Dirichlet distribution that sensors can use to update trust scores from direct interactions. In [C. J. Fung, J. Zhang, et al., 2011], they add the concept of *acquaintances* to break or establish dynamic relationships to improve the performance of a CIDS. In many works of Fung et al. , collusion cannot occur due to how sensors pass *messages* similar to how we measure reliability (see Section 7.1.1). In our work, we cannot discard the possibility of collusion due to the fact that the *message* passing system does not work in our scenario. This is because we assume that it is not always possible to directly interact with other CIDS sensors. Instead, we sometimes rely on indirect interactions. This might happen, for example, when CIDS sensors belong to different organizations.

Collusion resistant trust models have been proposed in different fields. Dwarakanath et al. [2017] proposed a collusion resistant mechanism that detects dishonest Internet of Things (IoT) devices that share incorrect trust scores. Dishonesty and collusion is detected by translating the trust scores into vectors and using a cosine similarity metric to measure deviations. Although this collusion detection

mechanism does not target CIDSs, it meets some of our requirements. However, the creation of trust vectors, in order to use the cosine similarity metric, is not easily transferable to the CIDS domain given our requirements. This is because CIDS sensors do not consider the trust relationships of other sensors to compute trust scores, i. e., transitive trust is not possible (otherwise colluders would have a way to defeat our system).

Trust can be incorporated into many levels of the CIDS architecture we reference (see Section 2.3.3). Gil Pérez et al. [2013] incorporate trust into the *global detection* component. The system assesses the trust score of sensors sending alerts to determine if an alert is taken or not into account. At the *membership management* level, C. Fung et al. [2011] uses trust scores to decide whether CIDS sensors are removed from the system. Our contribution, *Sphinx*, serves as a *membership management* mechanism that prevents dishonest sensors from affecting the operations of CIDSs.

7.4 SPHINX: A COLLUDER-RESISTANT TRUST MECHANISM

In this section, we present our proposed evidence-based trust mechanism coined *Sphinx*. We begin this section by describing our mechanism and its assumptions. We then describe how *Sphinx* computes trust scores from evidence of the reliability of sensors. We provide Table 7.1 to help the reader follow the notation used in this and subsequent sections.

7.4.1 *The Mechanism and its Assumptions*

Let us assume a CIDS with a set of n sensors $\mathcal{S} = \{S_1, \ldots, S_n\}$. We assign each sensor S_i a trust score $\tau_i^{(t)} \in [0, 1]$ at time t. The trust scores S_i express how reliable sensors are. Sensors may exchange local knowledge with each other[2]. After all sensors S_i share their information, at time t, sensors evaluate the reliability of each other and update trust scores $\tau_i^{(t)}$. To simplify notation, trust scores $\tau_i^{(t)}$ at time t are simply denoted as τ_i.

In a setting where reliability is rewarded, a sensor S_i should be interested in maximizing its reliability and, consequently, its trust score τ_i. To maximize its reliability, a sensor can go through the hardships of being highly available and accurate to receive good ratings. However, sensors could also consider being dishonest about the reliability of others to worsen or improve the trust score of others. Furthermore, dishonest sensors could secretly form coalitions to boost their own trust scores while degrading trust scores of others. We consider that each sensor S_i may be either honest or dishonest. Dishonest sensors

2 This work assumes that if sensor S_i shares its local knowledge, all sensors $S_j \in \mathcal{S}$ with $j \neq i$ receive the knowledge unaltered.

S_i	sensor i
n	number of sensors
$\tau_i = \tau_i^{(t)}$	trust score of S_i at time t
$\tau_i' \,/\, \tau_i''$	evidence- / reliability-based trust score of S_i at time t
$P_j^{(i)}$	Cartesian point $(x_{P_j^{(i)}}, y_{P_j^{(i)}})$ related to how S_j rates S_i
$x_{P_j^{(i)}} \,/\, y_{P_j^{(i)}}$	$x-$ / y-coordinate of $P_j^{(i)}$
$\mathcal{P}^{(i)}$	set of all points $P_j^{(i)}$ for $j \in [1, n]$ and $j \neq i$
$o_j^{(i)}$	evidence submitted by S_j with respect to S_i
K	number of cluster centers
$\mathcal{C}_1, \ldots, \mathcal{C}_K$	clusters or classes of credibility
$M_1, \ldots, M_K$	center points of clusters $\mathcal{C}_1, \ldots, \mathcal{C}_K$
$y_{M_1}, \ldots, y_{M_K}$	y-coordinate of $M_1, \ldots, M_K$
$\omega_j^{(i)}$	weight of Cartesian point $P_j^{(i)}$
$\pi_1, \ldots, \pi_K$	mixing coefficients of $M_1, \ldots, M_K$
$o_j^{(i)}$	reward or punishment of S_j given its ratings of S_i
α_1	reward and penalty values used to calculate $o_j^{(i)}$
α_2	number of subdivisions of $o_j^{(i)}$
$F(\tau_i)$	weight balancing function of trust scores τ_i
η	mixing coefficient of τ_i' and τ_i'' to create τ_i

Table 7.1: Summary of the notation used throughout this chapter

might act alone or form coalitions. We make the following three assumptions with respect to the behavior of honest and dishonest sensors.

- *Assumption 1*. Honest sensors report evidence as accurate as they can but make mistakes that follow a Gaussian distribution[3].
- *Assumption 2*. Dishonest sensors submit tampered evidence following a Beta distribution[3] and can choose to submit accurate evidence to confuse the system following a Uniform distribution[3].
- *Assumption 3*. When dishonest sensors collude, they may only belong to one coalition.

Sphinx aims at mitigating the effect of coalitions and single dishonest sensors under the last three assumptions. *Sphinx* achieves this by calculating two partial trust scores for each sensor S_i, termed evidence-based and reliability-based trust scores, and making a linear combination of the two to obtain trust score τ_i. The *evidence-based trust score* τ_i' for sensor S_i is calculated using the reliability (i. e., the evidence) that all other sensors S_j calculated of sensor S_i, for $j = 1, \ldots, n$ and $j \neq i$ (see Section 7.4.2). The calculation of the evidence-based

τ_i': *evidence-based trust score*

3 We discuss why this distribution is chosen in Section 7.5.1.

trust score is similar to what Bayesian models accomplish (see Section 7.3), except that what we process is weighted differently depending on the reputation of the source (see below). The *reliability-based trust score* τ_i'' depends on how reliable the evidence submitted by sensor S_i is, with respect to sensor S_j, where $j \neq i$ (see Section 7.4.3). In the calculation of the reliability-based trust score, unreliable evidence is detected and whoever submitter it is penalized. In Section 7.4.4, we describe how these two partial trust scores are merged to obtain the final trust score τ_i.

τ_i'': *reliability-based trust score*

7.4.2 *Evidence-based Trust Score*

This section details how *Sphinx* computes the evidence-based trust score τ_i' of sensor S_i taking into account evidence submitted by all the other sensors $S_j \in \mathcal{S}$ where $j \neq i$. The computation of τ_i' is performed in a two dimensional Euclidean space $D = 2$. In the following, we detail the steps *Sphinx* follows for the computations.

COLLECTING EVIDENCE *Sphinx* starts by collecting, as evidence, how each sensor rates the reliability of all other sensors. *Sphinx* stores a Cartesian point of the form $P_j^{(i)} = (x_{P_j^{(i)}}, y_{P_j^{(i)}})$ that relates to how sensor S_j rates S_i, where $j \neq i$. The first component of point $P_j^{(i)}$ is $x_{P_j^{(i)}} = \tau_j^{(t-1)} \in [0, 1]$, where $\tau_j^{(t-1)}$ is the last known trust score of the sensor S_j. The second component of point $P_j^{(i)}$ is $y_{P_j^{(i)}} = \sigma_j^{(i)}$, where $\sigma_j^{(i)} \in [0, 1]$ is how S_j rates the reliability of S_i. The rating $\sigma_j^{(i)} = 0$ indicates that sensor S_j has only seen S_i being unreliable; $\sigma_j^{(i)} = 1$ is the complete opposite (see Example 7.1 for possible ways to compute reliability). With all the evidence, we create datasets of the form $\mathcal{P}^{(i)} = \{P_1^{(i)}, \ldots, P_{i-1}^{(i)}, P_{i+1}^{(i)}, \ldots, P_n^{(i)}\}$. The dataset $\mathcal{P}^{(i)}$ represents all Cartesian points (evidence) that relate to the reliability of sensor S_i.

FORMING CREDIBILITY CLASSES *Sphinx* groups the submitted evidence into different classes, termed *credibility classes*, using clustering. K-means clustering groups the points of $\mathcal{P}^{(i)}$ into K clusters $\mathcal{C}_1, \ldots, \mathcal{C}_K$. Given that the points $P_j^{(i)} \in \mathcal{P}^{(i)}$ contains both the reputation of the submitter sensor S_j and the submitted evidence $\sigma_j^{(i)}$, k-means finds clusters that take into account both values. The clusters $\mathcal{C}_1, \ldots, \mathcal{C}_K$ contain points that rate S_i submitted by sensors with a similar reputation. From the resulting cluster centers $M_1, \ldots, M_K$, however, *Sphinx* only needs their y-coordinates $y_{M_1}, \ldots, y_{M_K}$ to categorize evidence into credibility classes. This is because y_{M_i} ends

credibility classes

up representing the average trust score submitted by the sensors in cluster $\mathcal{C}_i$ (recall the definition $P_j^{(i)} = (x_{P_j^{(i)}}, y_{P_j^{(i)}}) = (\tau^{(t-1)}, \sigma_j^i)$).

WEIGHTING EVIDENCE *Sphinx* uses the previous reputation $\tau_j^{(t-1)}$ of sensor S_j to weight the points $P_j^{(i)}$ that S_j submitted. The weight of a point is calculated with the function

$$\omega(P_j^{(i)}) = \frac{F(x_{P_j^{(i)}})}{\sum_{k=1}^{n-1} F(x_{P_k^{(i)}})}.$$

The function $F : [0,1] \to \mathbb{R}$ is a positive and increasing function over the interval $[0,1]$ meant to assign high values to high trust scores $\tau_j^{(t-1)} = x_{P_j^{(i)}}$. The purpose of this function is to balance the influence of low trust scores against high ones. In other words, function $F(x)$ determines how many low trust scores are needed to have enough influence to overcome high trust scores. This is desirable as sensors with high trust scores might also submit incorrect evidence. If many sensors, even with low trust scores, submit evidence that contradicts a sensor with high trust score, their opinions also have an impact. A possible approach to define function $F(x)$ is to choose a step function over disjoint sub-intervals of the interval $[0,1]$. A concrete instantiation of this function is later shown in the evaluation section (see Section 7.5).

MIXING EVIDENCE AND CREDIBILITY CLASSES The last step to calculate the evidence-based trust scores τ'_i of sensor S_i is to combine the previously defined credibility classes and the weighted evidence. The combination is done with the help of GMMs. A GMM is composed by a linear combination of Gaussian distributions $\mathcal{N}_i$ with corresponding mixing coefficients π_i (see Section 7.2.2). Each credibility class is used to represent a Gaussian distribution such that, for K credibility classes, we create the K Gaussian distributions $\mathcal{N}_1(\mu_1, \sigma_1^2), \ldots, \mathcal{N}_K(\mu_K, \sigma_K^2)$. The parameters of the Gaussian distributions are set so that $\mu_k = M_k$ (the center of cluster k) and σ_k^2 is the average square distance between all points in cluster $\mathcal{C}_k$ and cluster center M_k. To avoid σ_k^2 being zero, we add to it the smoothing term 10^{-5}. The mixing coefficients π_k are calculated using the weights of the evidence with

$$\pi_k = \sum_{l=1}^{\|\mathcal{C}_k\|} \omega(u_l^{(k)})$$
$$u_l^{(k)} \in \{P_j^{(i)} \mid P_j^{(i)} \in \mathcal{C}_k\}.$$

In summary, the weights of all points in cluster $\mathcal{C}_k$ are summed together to form π_k. Finally, with the Gaussian distributions and mixing

coefficients calculated, the GMM is approximated with a single Gaussian distribution[4]. The mean of the Gaussian approximation becomes our *evidence-based trust score* τ'_i of sensor S_i:

$$\tau'_i = \sum_{k=1}^{K} \pi_k \cdot \mu_k \in [0, 1].$$

7.4.3 *Reliability-based Trust Score*

This section covers how *Sphinx* computes the reliability-based trust score τ''_i. This partial trust score compares the reliability of the evidence submitted by sensor S_i regarding sensor S_j against all evidence sent by sensor S_k, for $k \neq i, j$, targeting S_j. The purpose of this computation is to distinguish reliable from unreliable evidence and to discourage tampered submissions. As in the calculation of τ'_i, the reliability-based trust score τ''_i is performed in a two dimensional Euclidean space. The following describes the computation in a series of steps.

PROCESSING EVIDENCE In this first step, we use the same evidence $\mathcal{P}^{(i)} = \{P_1^{(i)}, \ldots, P_{i-1}^{(i)}, P_{i+1}^{(i)}, \ldots, P_n^{(i)}\}$, where $P_j^{(i)} = (\tau_j^{(t-1)}, \sigma_j^{(i)})$, that is collected in the first step of the calculation of τ'_i. All submitted evidence $\sigma_j^{(i)} = y_{P_j^{(i)}}$ are compared against τ'_i using the Euclidean distance, denoted as $d(\tau'_i, \sigma_j^{(i)})$. When, $d(\tau'_i, \sigma_j^{(i)}) = 1$, the sensor S_j submitted evidence that contradicts the calculated evidence-based trust score of sensor S_i. In contrast, if $d(\tau'_i, \sigma_j^{(i)}) = 0$, sensor S_j submitted evidence that matches the calculated evidence-based trust score of sensor S_i. In the next step, contradictory and consistent information is, respectively, punished or rewarded.

ASSIGNING RELIABILITY SCORES *Sphinx* rewards or punishes sensor S_j based on its value of $d(\tau'_i, \sigma_j^{(i)})$ according to a set of dynamically discretized ranges. We term these discretized ranges *reliability scores*. We use $o_j^{(i)}$ to denote the reliability score assigned to sensor S_j for the rating it submitted of sensor S_i. The value of $o_j^{(i)}$ depends on how close it is to τ'_i. The values of $o_j^{(i)}$ lie in the set

reliability scores

$$\mathbf{O} = \{x \mid x = \alpha_1 - (\alpha_1 \cdot n);\ \forall n \in \{0, 1, 2, \ldots, (\alpha_2 - 1)\}\},$$

where the parameter $\alpha_1 \in [0, 1]$ is used to specify a series of rewards or penalty values and the parameter $\alpha_2 \in \mathbb{Z}^+$ specifies the total number of elements (steps) in $\mathbf{O}$. Intuitively, the first value of $\mathbf{O}$ is α_1, the

4 See Section 7.2.2 for a justification of why one Gaussian distribution is used as approximation.

second is 0 and subsequent ones are multiples of $-\alpha_1$, for a total of α_2 elements. *Sphinx* assigns $o_j^{(i)}$ to $d(\tau_i', \sigma_j^{(i)})$ by finding the element in $\mathbf{O}$ with index $\lfloor d(\tau_i', \sigma_j^{(i)}) \cdot \alpha_2 \rfloor$. For example, if distance $d(\tau_i', \sigma_j^{(i)})$ is between the range $\left[0, \frac{1}{\alpha_2}\right]$, the first element of $\mathbf{O}$, namely α_1, is assigned as $o_j^{(i)}$.

ALLOCATING PUNISHMENTS AND REWARDS As last step, *Sphinx* calculates the reliability-based trust score τ_j'' of each sensor S_j based on all reliability scores $o_j^{(i)}$, for $i = 1, \ldots, n$ and $j \neq i$, using

$$\tau_j'' = \tau_j^{(t-1)} + \frac{1}{n-1} \sum_{i=1, i\neq j}^{n} o_j^{(i)}.$$

τ_j'' is the result of increasing or decreasing the last trust score of sensor S_j by the average reliability scores of the sensor. With this calculation and the way reliability scores are calculated, reliability-based trust builds up slowly over some period of time. Conversely, it is lost quickly.

7.4.4 *Final Trust Score*

Trust score τ_i is a linear combination of τ_i' (Section 7.4.2) and τ_i'' (Section 7.4.3). Given the parameter $\eta \in [0, 1]$, we calculate the trust score τ_i of sensor S_i as:

$$\tau_i = \eta \cdot \tau_i' + (1 - \eta) \cdot \tau_i''. \tag{7.1}$$

The parameter η can be chosen based on the requirements of a specific CIDS. When η is close to one, the evidence-based trust score is favored over the reliability-based trust score.

7.5 EVALUATION

Sphinx is capable of identifying coalitions if less than 50 percent of all collaborating sensors collude within a single coalition. In certain conditions, when multiple and independent coalitions exist, *Sphinx* can identify dishonest participants even when more than 50 percent of all sensors are dishonest. This section provides experimental evidence to support these claims.

7.5.1 *Experimental Setup*

We perform all experiments in rounds. In each round t, *Sphinx* carries out all the operations needed to calculate the trust score $\tau_i^{(t)}$ for each sensor S_i in the CIDS. In summary, *Sphinx* calculates first the evidence-based trust score τ_i' followed by the reliability-based trust score τ_i''.

Afterwards, *Sphinx* combines these two partial trust scores into a final one using Equation 7.1

The sensors in all experiments are either honest or dishonest except when we explicitly indicate that this is not the case. An honest sensor rates others according to its empirical observations, making small mistakes that follow a Gaussian distribution. A dishonest sensor rates others better or worse, following a Beta distribution, depending on whether the sensor colludes with others or not. If a dishonest sensor is acting alone, without the support of others, it rates every other sensor worse than what it itself empirically observed. In doing so, the supposition is that his trust score would eventually become the best if the score of everyone else worsens. These single dishonest sensors are termed *lone defectors*. If a dishonest sensor colludes with other sensors to form coalitions, the following rules apply:

lone defector

1 When rating sensors in the same coalition, the submitted ratings are improved relative to the empirical observations of the rated sensor.

2 Sensors of a coalition rate sensors not belonging to the same coalition with a worsened rating relative to the true empirical observations. With some probability, dishonest sensors can give honest ratings to try and fool the system.

3 Dishonest sensors can only belong to one coalition.

4 Multiple coalitions may exist.

All non-deterministic experiments are repeated 50 times. Instead of showing aggregated graphs of all experiments, we show one scenario that represents the overall tendency of the experiments. *Sphinx* is a deterministic algorithm; the behavior of the sensors is not. The stochastic behavior of the sensors does not heavily modify the trend of our experiments; therefore, the variance of the results is small. For this reason, variance is not shown.

7.5.1.1 *The Parameters of the Rating System*

Dishonest sensors submit evidence tampered positively or negatively following a Beta distribution. In the next experiments, whenever dishonest sensor S_j submits a rating, or evidence, for S_i. It does so using

$$x \sim \mathrm{Beta}(a, b) \tag{7.2}$$

$$\sigma_j^{(i)} = \tau_i \pm x, \tag{7.3}$$

where a and b are parameters of a Beta distribution and use one of the combinations shown in Figure 7.2. In Equation 7.2, a sample x of the chosen Beta distribution is obtained. This sample is either added or subtracted, in Equation 7.3, depending on whether the true trust

score τ_i is improved or worsened. We choose the Beta distribution as it models dishonest members that alter evidence conservatively most of the time and aggressively a few times with low probability.

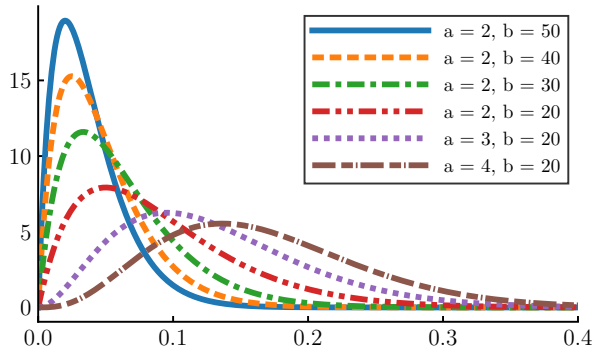


Figure 7.2: Family of Beta distributions that specifies how much ratings are improved or worsen by dishonest participants in a CIDS.

Honest sensor S_j submits ratings, or evidence, concerning another sensor S_i, following a Gaussian model; that is,

$$y \sim \mathcal{N}(0, \mathtt{std}) \tag{7.4}$$

$$\sigma_j^{(i)} = \tau_i + y. \tag{7.5}$$

The sample y is added to the true trust score τ_i to account for potential inaccuracies in the empirical observations of S_j. In all experiments, $\mathtt{std}$ is chosen such that $\mathtt{std} < \mathbb{E}[\mathtt{Beta}(a, b)]$. If this was not given, the samples of the Gaussian distribution used by honest sensors would greatly overlap with the samples of the Beta distribution used by dishonest sensors, making the behavior of honest and dishonest sensors almost indistinguishable.

7.5.1.2 *The Parameters of* Sphinx

The calculations of *Sphinx* depend on different user-supplied parameters. The evidence-based trust score τ' uses two parameters: a function $F(x)$ that weights scores and the number of cluster centers K (see Section 7.4.2). The reliability-based trust score τ'' relies on two parameters: a reward parameter α_1 and the number of penalty subdivisions α_2 (see Section 7.4.3). The final trust score τ depends on a mixing coefficient η (see Section 7.4.4). In most experiments, all these parameters are fixed to a single set of values.

7.5.2 *Experiments*

We conduct experiments to demonstrate how the trust scores of lone defectors and coalitions are penalized. Unless explicitly indicated, all experiments use the following parameters:

$$F(x) = \begin{cases} 1 & \text{if } 0.00 \leqslant \tau \leqslant 0.25 \\ 2 & \text{if } 0.25 < \tau \leqslant 0.50 \\ 3 & \text{if } 0.50 < \tau \leqslant 0.75 \\ 4 & \text{if } 0.75 < \tau \leqslant 1.00. \end{cases},$$

$K = 2$, $\alpha_1 = 0.20$, $\alpha_2 = 25$ and $\eta = 0.30$. Honest sensors use $\mathrm{std} = 0.05$ in Equation 7.4. Dishonest sensors use the parameters $a = 3$, $b = 20$ for the Beta distribution in Equation 7.2. These parameters for the Beta distribution yield the expected value of $\mathbb{E}[\mathrm{Beta}(3, 20)] = 0.13$. That is, most times, dishonest sensors alter (positively or negatively) the rating of others by 0.13 points. The rating alterations generally range from values close to 0 up to 0.35. That is, the chosen Beta distribution models dishonest sensors that choose to be conservative most times but aggressive a some others.

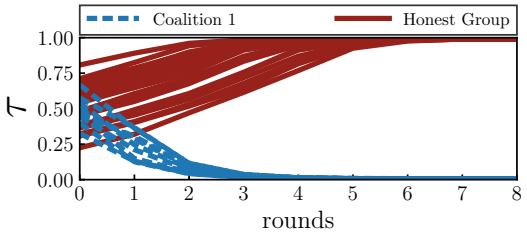
7.5.2.1 *Experiment 1: Detecting Single Coalitions*

Sphinx can detect single large coalitions as long as the number of sensors in that coalition do not exceed the number of honest sensors. This is shown in Figure 7.3. The x-axis of each plot shows the number of rounds performed by *Sphinx*. Round zero is the case where *Sphinx* has not run and represents the initial bootstrapped trust of each sensor. All sensors are bootstrapped with an initial trust score following the Gaussian distribution $\mathcal{N}(0.50, 0.15)$. On the y-axis, the trust scores τ are shown.

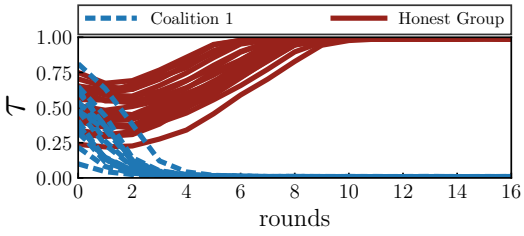
In the four plots of Figure 7.3, different coalition sizes are tested to see how the trust scores of all sensors are affected. Figure 7.3a shows that our mechanism successfully and quickly (in three rounds) lowers the trust score of all members in a coalition made up of 25 percent of all sensors. Figure 7.3b and 7.3c show that, although the coalition has some influence for a few rounds, the trust score τ of the coalition members eventually falls to zero. Notice that in Figure 7.3c 47.5 percent of all sensors (i. e., 19 out of 40) are dishonest. If 50 percent of all sensors are part of a coalition, *Sphinx* fails to punish the coalition, as Figure 7.3d shows.

7.5.2.2 *Experiment 2: Detecting Multiple Coalitions*

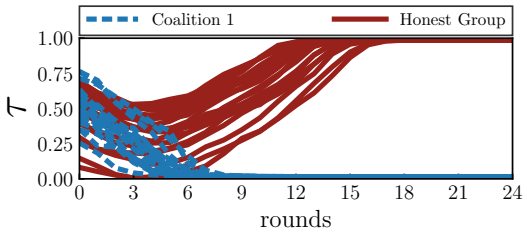
Sphinx can recognize and punish multiple independent coalitions. With multiple coalitions, even if more than 50 percent of the total



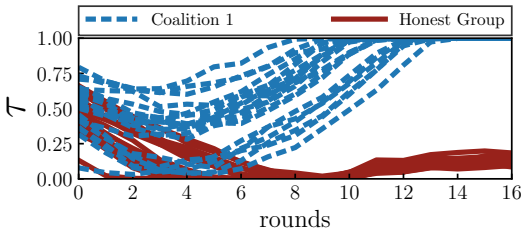
(a) 40 sensors in total, 25% of all sensors are dishonest. The colluding sensors are easily identified; their trust rapidly drops to zero.



(b) 40 sensors in total, 37.5% of all sensors are dishonest. In the first two rounds, the coalition can negatively affect the honest sensors. After the trust of the coalition drops to zero, the honest sensors only increase their trust.



(c) 40 sensors in total, 47.5% of all sensors are dishonest. Same effect as when considering 37.5% of dishonest sensors. With more dishonest sensors, the effect last longer until about round six.

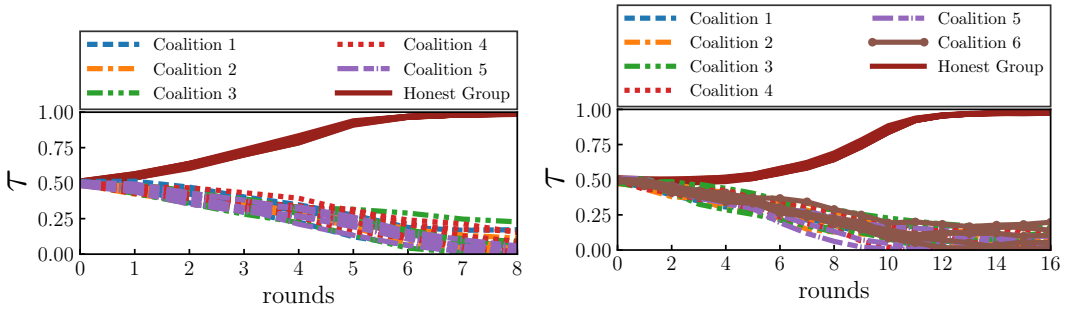


(d) 40 sensors in total, 50% of all sensors are dishonest. The dishonest sensors overwhelm the honest sensors and they easily drop the rating of the honest sensors to low values.

Figure 7.3: Change of trust with every new calculation (round) of *Sphinx*. Four different scenarios are evaluated: when 25%, 37.5%, 47.7% and 50.0% of the collaborating sensors form a dishonest coalition.

sensors are dishonest, honesty is successfully rewarded and dishonesty punished. In Figure 7.4, we see the good performance of *Sphinx*. In Figure 7.4a, a scenario with five coalitions is tested, each coalition having 5 sensors; amounting to 50 percent of all sensors. Similarly, in Figure 7.4b, a scenario of six coalitions with 5 sensors each is tested. Notice that in the last scenario 60 percent of all sensors are dishonest (although not from the same coalition). Honesty is successfully rewarded while dishonesty is punished.

When dishonest sensors use our default Beta distribution with parameters $a = 3$ and $b = 20$, adding more coalitions (of 5 sensors) would result in an ecosystem were no individual or coalition can increase its trust score beyond 0.25. Dishonest sensors effectively deny the possibility of gaining trust. If the dishonest sensors are less conservative and modify their ratings using a more aggressive Beta distribution with parameters $a = 2$ and $b = 10$, honesty is still recognized and rewarded as shown in Figure 7.5. The figure shows that during the early rounds, all trust scores are heavily penalized. After the trust scores stop fluctuating, honesty is recognized again. In round 12, dishonest sensors no longer have enough trust and their dishonest sub-



(a) 50 sensors and 5 coalitions of 5 sensors each. 50% of all sensors are dishonest. Although 50% of the total sensors are dishonest, the trust scores of dishonest sensors rapidly drop to zero. The trust score of the honest sensors rapidly increases.

(b) 50 sensors and 6 coalitions of 5 sensors each. 60% of all sensors are dishonest. Dishonesty is identified despite having more dishonest sensors overall. Notice that double the rounds were needed to detect dishonesty in contrast to (a).

Figure 7.4: Change of trust with every new calculation (round) of *Sphinx*. Four different scenarios are evaluated: when 25%, 37.5%, 47.7% and 50.0% of the collaborating sensors form a dishonest coalition.

missions stop having much weight. It is in this moment that honest sensors slowly build up trust once again.

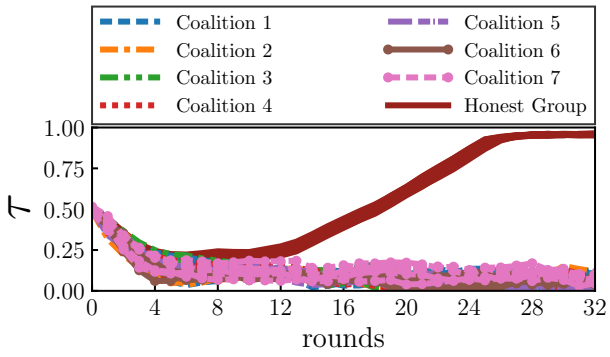


Figure 7.5: 50 sensors and 7 coalitions of 5 sensors each. 70% of all sensors are dishonest. If the dishonest sensors are less conservative, honest sensors are eventually recognized.

7.5.2.3 *Experiment 3: The Effects of Dispersed Bootstrapped Trust Scores*

The previous two experiment assumed that all sensors start with a bootstrapped trust score close to 0.5. Initializing the trust scores over the range $[0, 1]$ has no negative influence on the capabilities of *Sphinx* to detect dishonesty. Taking into account 20 honest sensors and two coalitions of 10 sensors each (for a total of 20 dishonest sensors), Figure 7.6 shows how honest sensors with a low bootstrapped trust score eventually reach high trust values. Similarly, dishonest sensors with high bootstrapped trust scores get their score reduced two zero given

enough rounds. As shown in the figure, the honest sensor with the lowest starting trust score (of 0.08) is able to reach a maximum score in eleven rounds. The dishonest sensor with the highest initial trust score (of 0.95) is reduced to a score of zero after ten rounds.

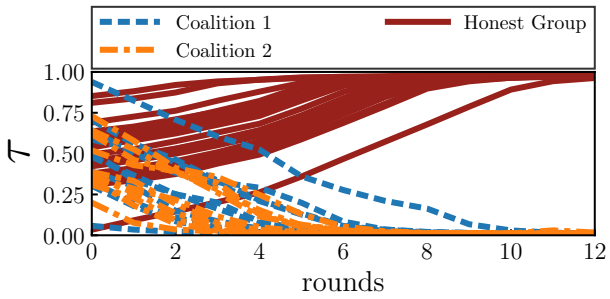


Figure 7.6: Evolution of the trust scores (τ) of two coalitions, each with 10 sensors, and 20 honest sensors when the trust scores are initially dispersed. The honest sensor with the lowest score (of 0.08) reaches a trust of 1.00 after 11 rounds.

7.5.2.4 *Experiment 4: The Effects on the Sensibility of Dishonesty*

We examine how varying the sensibility of dishonesty affects the capability of *Sphinx* to identify coalitions. In the following experiment, the trust scores of sensors are bootstrapped following the Gaussian distribution $\mathcal{N}(0.5, 0.2)$. This initialization simulates that all sensors start with similar trust scores. In our experiments, we observe that honesty is identified when $\mathtt{std} < \mathbb{E}[\mathtt{Beta}(a, b)]$. Thereafter, we choose in our experimental setup (Section 7.5.1) $\mathtt{std} = 0.05$. With this parameter value, honest sensors approximate the real score τ_i of sensor S_i with $\hat{\tau_i}$ such that $\hat{\tau_i} = \mathcal{N}(\tau_i, 0.05)$.

From the illustrated Beta distributions in Figure 7.2, *Sphinx* can detect dishonest sensors that act according to a Beta distribution parameterized with $a = 2$ and $b = 30$ (where $\mathbb{E}[\beta(2, 30)] = 0.062$) and below. Conversely, if dishonesty is modeled with the Beta distribution $\beta(2, 40)$ or $\beta(2, 50)$, dishonesty cannot be identified as it is easy to confuse with honest mistakes.

Figure 7.7 shows the average trust scores τ with standard deviations after executing 12 rounds of *Sphinx*. When the coalitions are dishonest, following the Beta distribution $\beta(2, 20)$ and $\beta(2, 30)$, the average trust score of the dishonest participants is kept low. In all repeated experiments, the average trust score of all coalitions tends to 0. This is not the case, however, when coalitions act according to the Beta distributions $\beta(2, 40)$ or $\beta(2, 50)$. In both these cases, the average trust scores cannot be kept low and have a tendency to increase towards 1. This is due to the fact that with such Beta distributions, it is not possible to distinguish dishonesty from honest mistakes. Note

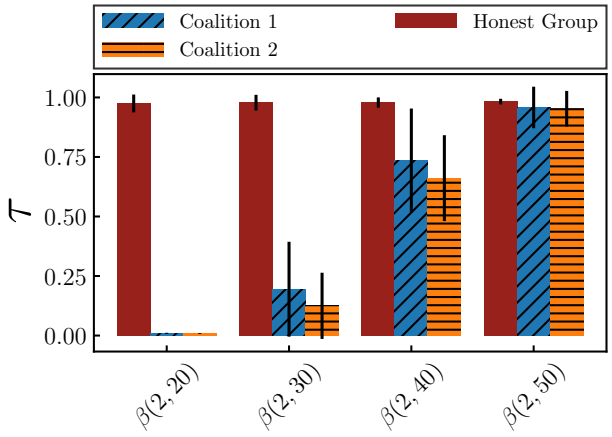


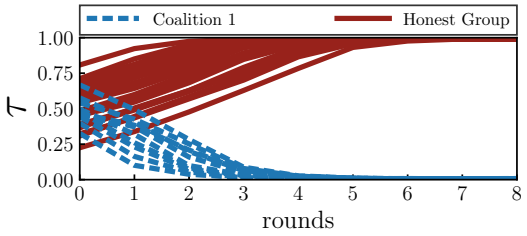
Figure 7.7: Average trust score τ, after 12 rounds, of honest and dishonest sensors when the sensibility of dishonesty changes. From left to right, more subtle Beta distributions are used by dishonest sensors to improve or worsen the trust score of others. When using $\beta(2, 30)$ or $\beta(2, 20)$, the trust scores of both coalitions are kept in line. Using more subtle Beta distributions results in the trust scores of the coalitions also increasing (without affecting the honest participants).

that the trust scores of the honest sensors are not negatively affected this way.

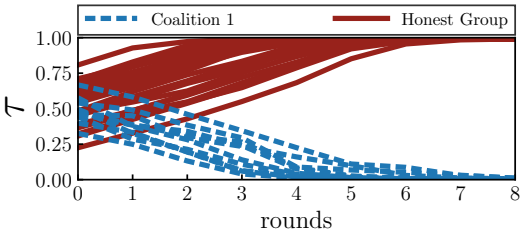
7.5.2.5 *Experiment 5: Dealing with Smarter Dishonest Sensors*

In this scenario, we describe the effects on honest and dishonest sensors when dishonest ones choose to sometimes submit honest evidence according to some probability p to fool *Sphinx*. In Figure 7.8, we illustrate four scenarios that take into account 40 sensors, different ratios of dishonest sensors, and different values of p. Subfigure 7.8a and 7.8b duplicate the conditions and setup of the experiments shown in Figure 7.3a but incorporate p. In Figure 7.8a, each sensor of the coalition chooses to be honest with a probability of 20 percent ($p = 0.2$). In contrast to the results obtained when $p = 0$ (cf. Figure 7.3a), the trust scores of the coalition stay slightly higher in round two but almost collapse by round three. With $p = 0.5$, as shown in Figure 7.8b, the trust scores of the coalition are slowly punished. In the last two scenarios, the trust scores of the honest are barely affected.

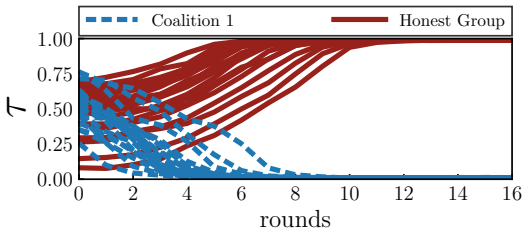
Figure 7.8c and 7.8d duplicate the conditions and setup of the experiments shown in Figure 7.3c but incorporate p. As shown in Figure 7.8c, with $p = 0.2$, the coalition is not successful in keeping their trust scores relevant. By round eight, all trust scores collapse. Surprisingly, the trust scores of honest sensors are positively rewarded by the honest submissions of the coalition. The trust scores of honest sensors do not decrease as they do in Figure 7.3c and are maximized by round 11. In the experiment shown in Figure 7.8d, sensors in the



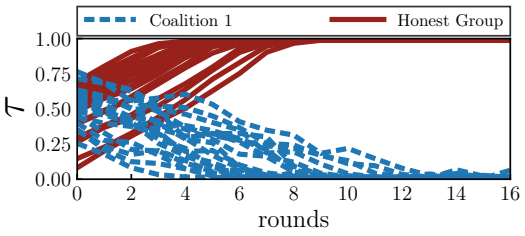
(a) 40 sensors, 25% dishonest, 20% probability of dishonest sensors being honest. When colluders are 20% of the time honest, honest sensors increase their trust scores faster. Dishonest sensors are still identified.



(b) 40 sensors, 25% dishonest, 50% probability of dishonest being sensors honest. Colluders are correctly identified, albeit in later rounds.



(c) 40 sensors, 47.5% dishonest, 20% probability of dishonest sensors being honest. When sensors in coalition 1 rate honestly with a 20% probability, instead of harming the honest sensots, the coalition boosts the trust scores of the honest members without gaining enough trust.



(d) 40 sensors, 47.5% dishonest, 50% probability of dishonest sensors being honest. With a 50% probability of rating honestly, coalition 1 further boosts the scores of the honest sensors. Their trust scores, however, are more slowly reduced.

Figure 7.8: The performance of *Sphinx* when dishonest sensors are smarter, i. e., they act honestly with some probability. The top and bottom rows replicate the setup used in the first experiment, as shown in Figure 7.3a and 7.3c, respectively, but consider smarter dishonest sensors.

coalition choose to be honest 50 percent of the time. In this scenario, the trust scores of the coalition stay relevant longer but eventually collapse after enough rounds pass. However, the trust scores of the sensors are positively affected and reach their maximum earlier by round eight.

Overall, dishonest sensors that choose to act honestly with some probability p must trade between staying relevant longer and rewarding honest members to increase their score even faster. In the scenario of smart dishonest sensors, we found a turning point when $p = 0.65$, for which the results are shown in Figure 7.9. At this point, sensors in coalition 1 choose to be honest 65 percent of the time and manage to keep their trust scores almost constant for all 32 rounds. If the coalition wishes to get the trust scores τ of their sensors to rapidly increase (and not be left behind by the honest sensors), they would need to be honest 80 percent of the time (not shown in the figure). This would enable them to give some dishonest ratings without be-

ing heavily punished but would go against the goal of reducing the trust scores of honest sensors.

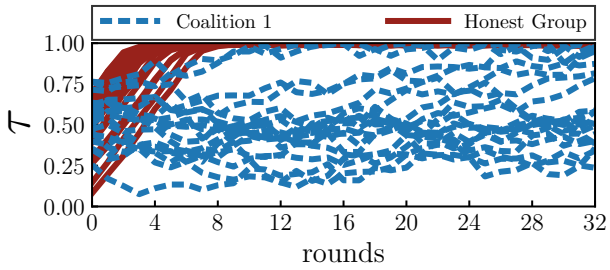


Figure 7.9: 40 sensors, 47.5% dishonest, 65% probability of dishonest sensors being honest. The trust scores of the coalition almost stay constant throughout 32 rounds of running *Sphinx*. Yet, the coalition could be identified by the fact that their trust scores do not increase as those of the honest sensors.

7.6 CONCLUSION AND LESSONS LEARNED

This chapter proposed *Sphinx*, an evidence-based trust mechanism that can detect dishonest sensors within a CIDS. *Sphinx* collects the evidence submitted by all sensors regarding the reliability of other sensors. From this evidence, each sensor is assigned a trust score that represents the belief of the system about the attitude of the sensor. Dishonest sensors can be identified using the iterative computations of *Sphinx* in just a few rounds.

Related work proposes diverse evidence-based trust mechanisms to solve the issue of detecting dishonest CIDS members. Related work often makes assumptions, however, that simplify the problem to better adjust it to known computational trust models. As a goal of this work, instead, we aimed at developing an evidence-based trust mechanism that can cope with *smart* dishonest sensors and the initial bootstrap problem. Smart dishonest sensors are sensors that behave inconsistently overall, i. e., they switch roles. Inconsistent behavior is known to cause troubles when designing evidence-based trust mechanisms [Burnett et al., 2010]. We relaxed the "smartness" assumption and designed our mechanism so as to assume that dishonest sensors choose to be honest with some probability.

Evidence-based trust mechanisms tend to rely excessively on their capability to bootstrap trust scores reasonably well. If the initial trust score of a sensor is incorrectly bootstrapped, an honest sensor might struggle to gain trust for a long time while dishonest sensors might benefit for far too long [Burnett et al., 2010]. We therefore focused on designing a mechanism that does not heavily dependent on the initial trust score of sensors. We achieve this with the help of clustering algorithms and Gaussian Mixture Models (GMMs). We were able to show in our evaluations that bootstrapping all trust scores to a fixed

middle value (0.5) or uniformly within a wide range ([0.25,0.75]) does not significantly affect *Sphinx* negatively.

7.6.1 *Future Work*

As future work, we intent to extend *Sphinx* to compute trust scores incorporating additional criteria such as the amount of past interactions between sensors, the initial bootstrapped trust score, and the rate of change of trust scores. *Sphinx* is already capable of processing additional information thatn the one that we already process through the usagage of clustering to define *credibility classes*. However before new criterias can be introduced, they need to be transformed to a two dimensional Euclidean space that matches the one *Sphinx* uses. We further plan to design a new bootstrapping procedure that improves upon [Xin Liu, Datta, et al., 2009] to better address the scenario of trust within CIDSs.

7.6.2 *Chapter Summary*

Honesty is a trait that should not be assumed to be part of all CIDS members. This is especially true when the sensors of multiple independent organizations collaborate while having potentially different goals and purposes. In this chapter, we presented *Sphinx*, an evidence-based trust mechanism that uses the reliability of sensors to compute an overall trust score for each sensor. In *Sphinx*, the trust score of a sensor represents the believe of the system concerning the attitude of the sensor. Sensors with high trust scores are thought to be honest, i. e., they share the overall goal of the CIDS. Low trust scores are exactly the opposite and point towards dishonest sensors.

Sphinx makes three assumptions regarding how sensors submit evidence about the reliability of others. First, it assumes that honest sensors make small mistakes that follow a Gaussian distribution. Second, dishonest sensors submit tampered evidence with some probability. Dishonest sensors tamper evidence (positively or negatively) following a Beta distribution. Third, dishonest sensors can coordinate into groups to collude; however, one sensor can only belong to one such group. In contrast to related work, as implied by the second assumption, dishonest sensors can sometimes be honest to try and fool the system.

Sphinx uses computational trust models and unsupervised ML algorithms to compute the trust scores of sensors. The final trust score of a sensor depends on the two other partial trust scores *evidence-based trust score* and *reliability-based trust score* that *Sphinx* calculates. In the evidence-based trust score, *Sphinx* uses all the collected evidence that refers to one sensor to calculate its first partial trust. In the reliability-

based trust score, the calculations done in the evidence-based trust score are compared against all submissions to identify outliers.

We conduct five experiments to test the performance and weaknesses of *Sphinx*. The first experiment shows how *Sphinx* can detect single large coalitions (groups of coordinated dishonest sensors) as long as they do not exceed the number of honest sensors. Experiment two shows how multiple coalitions can also be correctly identified when the largest coalition has no more members than the total number of honest sensors. This considers the scenario where the total number of dishonest servers is larger than the honest ones. In this scenario, dishonesty can still be punished. Experiment three tests the impact on trust scores when sensors start with diversified trust scores. Even in the presence of dishonest sensors with high trust scores and honest sensors with low trust scores, dishonesty can still be identified. In Experiment four, we test the limits of identifying dishonesty when evidence is tampered with different degrees of subtlety. Dishonest members need to be so subtle to not be detected that it overlaps with genuine errors made by honest sensors. Finally, in Experiment 5, we show how dishonest sensors that choose to sometimes be honest can be identified if they are dishonest more than 65 percent of the time.

8 CONCLUSION

Throughout this thesis, we introduced novel methods and mechanisms that enhance the capabilities of distributed Collaborative Intrusion Detection Systems (CIDSs) with the help of Machine Learning (ML). We propose solutions to problems that relate to one or many components that make up the architecture of a CIDS. In this section, we first summarize each chapter of this thesis, resuming how ML is used, what our contributions are, and which component of the CIDS architecture are affected. We then organize our conclusions in relation to the different components of the CIDS architecture. Finally, we provide an overlook of research areas and directions towards further improving the capabilities of distributed CIDSs.

8.1 SUMMARY

This thesis is composed of eight chapters, all relating to the common topic of improving CIDSs using aspects or theory borrowed from ML. The introduction chapter together with the related work and background chapter make up the *preface* of this thesis. The preface highlights the importance of CIDSs in large networks and covers the necessary topics needed to understand our contributions. The *core* of the thesis is composed of five chapters of contributions. Each of these core chapters explains and tackles an open issue within CIDSs. Overall, with the algorithms, systems and concepts proposed in our contributions, we enable the development of CIDSs that have better distributed capabilities.

thesis preface

thesis core

In the introduction, Chapter 1, we motivate the need and emphasize the challenges of detecting attacks in large networks. Distributed CIDSs are among the tools that precisely aim at solving this problem. However, in their current form, distributed CIDSs have disadvantages that constrain their ability to function adequately. For distributed CIDSs to function in practice rather than only in theory, they need to overcome diverse challenges. The introduction discusses these open challenges and presents our research objectives aimed at overcoming them.

Chapter 1

In the background and related work, Chapter 2, we present three key topics that relate to every core chapter alike. Specialized topics, those only relevant to a particular contribution, are covered within their respective chapter. The main covered topics of this chapter are ML, Network Intrusion Detection Systems (NIDSs) and CIDSs. From among the covered topics, we stress the importance of the section cov-

Chapter 2

ering the CIDSs architectural components (Section 2.3.3). Every contribution chapter uses the CIDS architecture described in this section to position the contribution within a CIDS. This helps the reader identify the area of a CIDS being improved.

Chapter 3

The first contribution of the thesis, detailed in Chapter 3, carries the name *Dataset Generation*. We begin the chapter describing the problems that the field of NIDSs and CIDSs have with regards to datasets. Datasets are key tools needed to design, evaluate and compare systems that rely on ML. The network intrusion detection field, however, notably lacks public datasets of quality. This can be attributed to many facts; the most important one being that datasets need to be labeled with ground truth. In a survey we present, we highlight how modern datasets with ground truth do not exist. To alleviate this issue, we propose the Intrusion Detection Dataset Toolkit (ID2T). ID2T creates labeled datasets with ground truth by generating synthetic attacks and injecting them into background traffic provided by the user. The synthetic attacks replicate the statistical properties (when appropriate) of the background traffic. This makes synthetic traffic difficult to distinguish from real traffic.

ID2T replicates the statistical properties of arbitrarily supplied background traffic to create synthetic attacks. Thereafter, the statistical quality of the background traffic has substantial impact on the synthetic traffic ID2T generates. We implement diverse metrics within ID2T to analyze network traffic to help identify when it contains irregularities or potential issues. For example, by measuring the distribution of new source IPs contacted in different time window, we can tell if traffic shows characteristics of a home, office or backbone network. We demonstrate the usefulness of the metrics by analyzing public datasets with known defects and showing which metric would potentially identify the defect.

The first contribution concludes with two use cases that demonstrate how ID2T can be used to test deployed NIDSs or evaluate new theoretical NIDSs. In the first use-case, we reproduce some of the evaluation results of an anomaly NIDSs we develop in Chapter 4. The second use-case shows how ID2T is used to validate two popular misuse NIDSs.

Chapter 4

The second contribution of this thesis, which is presented in Chapter 4 and titled *Intrusion Detection*, is the development of an unsupervised anomaly-based NIDSs aimed at working on backbone networks. Traditional NIDSs, whether distributed or not, cannot directly cope with the traffic produced in backbone networks because of two main reasons. One, backbone networks produce overwhelming amounts of network traffic and, two, creating intrusion detection models is challenging. We approach this problem by reducing the dimensionality of network traffic in multiple steps. First, packets are grouped into network flows. Second, network flows are characterized with the entropy

of many of their features. Finally, we use an autoencoder, known as a Replicator Neural Network (RNN), to reduce the dimensionality of the entropies. We use the RNN of the last step as the normality model of an anomaly detection system. With our system, we can detect undesired network-wide events, such as Distributed Denial of Service (DDoS) attacks and port scans, with low false positives. The performance of our system does not degraded when the traffic used to create a normality model contains attacks. We evaluate our system by using ID2T to injecting attacks into real network background traffic. Our evaluations show that we can identify attacks that leave relatively small traces of packets in relation to the massive quantity of total packets.

In our third contribution, with the name *Community Formation* and presented in Chapter 5, we develop concepts to form communities of NIDS sensors that enable us to apply centralized ML algorithms within distributed environments. A community is a group of sensors that collect network traffic in one location from where centralized models of normality can be learned. Communities distribute their learned models to all other communities. To detect intrusions, a community uses its model along with all others models it received (from other communities) to build an ensemble of models. We explore the effects of using different number of communities, communities of different sizes, and communities with different NIDS overlap. Our experiments show how communities leverage detection accuracy and communication overhead depending on their parameters. *Chapter 5*

Our fourth contribution is titled *Intrusion Information Dissemination* and is detailed in Chapter 6. In this chapter, we develop a mechanism tailored to the dissemination of information within a CIDS. CIDSs have special needs that typical information dissemination mechanisms (e. g., flooding or gossiping) do not adequately satisfy. For example, CIDSs need to distribute information with the guarantee that every member receives it. At the same time, members need to work with partial information without having to wait for the dissemination mechanism to converge. This last point is critical in large systems where dissemination may take a long time or when membership churn rate[1] is high. *Chapter 6*

Our dissemination mechanism finds to couple the Sketch Probabilistic Data Structure (PDS) and Bayesian Networks to lower communication overhead and enable CIDS members to make deductions from partial information. CIDS members encode the data (e. g., features) they want to distribute using Sketches. Sketches are then shared with neighboring members only. After receiving a Sketch, a member aggregates the data into the Sketch that it wishes to disseminate and forwards the Sketch to another neighbor. Sketches do not grow in size

1 The term "churn rate" is a metric that defines the rate by which individuals (e. g., sensors) enter or leave a group (e. g., a CIDS) over a period of time.

when data is aggregated in them. However, due to how they encode data, we loose the possibility of determining which member added which data to the Sketch (we only know which members participated in the aggregation process). Members use Bayesian networks to reason about the contents of a Sketch (i. e., what data belongs to what member). After seeing many Sketches, a Bayesian network can deduce what is it that each member placed in the Sketch. The deductions may later be used by a CIDS to identify distributed attacks. Our mechanism uses 20 percent less information than network flooding and can provide accurate deductions as soon as it sees one disseminated message.

Chapter 7

In our fifth and final contribution, detailed in Chapter 7 and titled *Collusion Detection*, we develop a system capable of detecting a single dishonest sensor or a group of colluding sensors within a CIDS. Our system exploits computational trust models and ML to compute trust scores of participating CIDS members that reflect their intentions. Our system uses the reliability of a member to compute a trust score. Reliability is the ability of sensor to reply correctly to queries in time and form. The correctness of a reply is established by hiding queries with known answers with every other query that a CIDS may ask one of its members. To determine the capabilities of our system, we conduct diverse evaluations that test different system conditions. Our system is capable of detecting coalitions (i. e., groups of colluding sensors) successfully as long as a colluding group is smaller than the group of honest sensors. Detecting dishonesty is possible even when there are more overall dishonest sensors than honest hones, given that no single colluding group is larger than the honest group. In contrast to related work, our system considers the possibility of dishonest sensors that choose to be honest (with some probability) to fool the system. These "smart dishonest sensors" do not affect the performance of our system and can be successfully detected.

8.2 ON THE USEFULNESS OF THE CONTRIBUTIONS

contributions are standalone

When all our contributions are put together, we can conceptualize a CIDS with improved distributed capabilities. However, our contributions are worthwhile by themselves and can improve the capabilities of existing CIDSs. With ID2T (Chapter 3), researchers can accurately compare CIDSs and ultimately design better ones. With our intrusion detection methodology based on RNNs (Chapter 4), CIDSs can adapt to detect undesired distributed traffic in both small and large networks. If centralized NIDSs already work well in centralized environments, our community creation concepts (Chapter 5) enable those NIDSs to be deployed within distributed environments by trading their detection accuracy for a desired communication overhead. A fully working distributed CIDS can replace its dissemination mechanism with the one

we propose (Chapter 6). This replacement enables CIDS members to deduce information even before they fully observe everything while, at the same time, reducing communication overhead (assuming the popular flooding mechanism was used). Regardless of the architecture a CIDS uses, with our collusion detection mechanism (Chapter 7), a CIDS can better protect itself from insider threats, enabling it to openly accept more collaborators with less risk. In the following, we match together the architectural components of CIDSs with our contributions and draw conclusions.

component enrichment

From the perspective of the CIDS architecture we reference (see Section 2.3.3), our contributions enrich each of the components that make up a CIDS. Let us consider the five components of a CIDS as shown in Figure 2.6 (p. 29). At the *local detection* component, we develop a dataset creation toolkit (Chapter 3) and an intrusion detection mechanism (Chapter 4) that improve the capability of individual CIDS sensors to detect intrusions. The intrusion detection mechanism we propose in Chapter 4 specializes on detecting anomalies in large networks. There are, however, no theoretical constraints that prevents it from performing well in smaller networks. Our dataset generation toolkit provides the means to test and design better systems that use ML. Many researchers argue that synthetic datasets are not as good as a real dataset [Abt et al., 2013]. However, we sympathize with the hypothesis that Abt et al. [2013] draw: If statistical learners (i. e., ML algorithms) are trained with synthetic datasets that replicate the statistical properties of real network traces, the probability is high that the learner can perform well in real-world settings.

local detection

membership management

Managing members in a CIDS is a challenging task. In real world settings, where multiple organizations may be involved in the detection of distributed threats (e. g., as in the PROTECTIVE H2020[2] project [*PROTECTIVE: Proactive Risk Management* 2018]), it is impossible to keep track of the actions of every collaborator. Trust therefore plays an important role in the process of managing members. Our evidence-based trust mechanism proposed in Chapter 7 can enable a more carefree environment for collaboration. Our trust-mechanism assumes that honest sensors make mistakes following a Gaussian distribution while dishonest ones follow a Beta distribution to tamper evidence. Although this assumption adequately cover most settings, we wonder how our system performs in settings where mistakes or malicious activities have different costs.

data dissemination

From an analysis of related work [Vasilomanolakis, Karuppayah, et al., 2015], we can conclude that the most neglected component of the five architectural components of a CIDS is the *data dissemination* one. We argue that this stems from the fact that disseminating data

2 Horizon 2020, also known as H2020, is a research program of the European Union with a funding close to €80 billions from 2014 to 2020. The program focuses on funding research that involves academic and industrial partnerships.

in a network is an already solved problem and, therefore, an engineering rather than a scientific task. In Chapter 6, on the contrary, we argue that disseminating information within a CIDS has special requirements that are not typically taken into account. For example, a system that analyses a network channel which it also uses for transportation, needs to be especially conscious of network overhead. By adding traffic to the same network it monitors, the system introduces potential bias in the detection process (e. g., due to congestion). If a countermeasure is implemented to ignore certain traffic, we may give malicious insiders the possibility of disguising themselves. Therefore, reducing communication overhead is crucial. At the same time, however, guaranteeing full information dissemination to achieve the est possible detection accuracy is important. These two goals are usually traded in regular dissemination techniques (e. g., gossiping [Kermarrec et al., 2007]).

data correlation and aggregation

The component that performs *data correlation and aggregation* is arguably the most important component of a CIDS. It is responsible for, both, reducing the amount of data that need to be transferred in a network and enabling the detection of distributed attacks experienced by different collaborators. The distribution mechanism we propose in Chapter 6 has the goal of fully distributing information while reducing communication overhead as much as possible. The mechanism uses aggregation and correlation as means to achieve its goals. Our mechanism therefore fuses two CIDS components together: those of *data dissemination* and *data correlation and aggregation*. We argue that CIDSs need to consider both components in conjunction to truly exploit the capabilities of aggregation, correlation and data dissemination.

The main goal of a CIDS is to detect network-wide threats. This is achieved by analyzing and observing traffic patterns that concern a network as a whole. In traditional CIDSs, those patterns come in the form of alarms. Alarms flow from the lowest component (i. e., *local detection*) all the way to the highest component (i. e., *global detection*), with no other information being shared. As a consequence, CIDSs detect network-wide threats only when the threats leave traces that trigger alarms. We have created the term *alarm level* to refer to CIDSs that only operate with alarms. Detection techniques at this level exist and can find distributed threats. We argue, however, that only using alarms is fundamentally flawed: The purpose of heavily distributed attack is to become subtle enough to avoid raising alarms.

global detection

Our contributions to the *global detection* component work at a different level that the alarm one. Instead of relying on alarms, we work on the foundation that network features can be shared to build distributed intrusion detection models. We refer to this level of CIDS operation as *detection level*. In Chapter 5, we propose the *detection level* and develop concepts to build CIDSs that operate at such level. The intru-

sion detection system we propose in Chapter 4, although presented as a single NIDS, can be implemented as the core detection strategy of a CIDS that operates at the detection level. Our contribution to develop labeled datasets also has an effect on the global detection component. With our synthetic datasets, we can train and test the performance of systems that try to detect network-wide threats.

8.3 OUTLOOK

In the course of this thesis, we presented diverse contributions that together improve modern CIDSs. Our approach consisted in recognizing the basic components that make a CIDS and providing improvements as a whole, or proposing solutions to outstanding issues. We envision further research in relation to each of our components but also to the CIDS in general. Each contribution chapter (i. e., from Chapter 3 to 7) includes a discussion of possible future work that relates to the contribution itself. In this section, we propose future work outside of our contributions to improve CIDSs.

We have previously argued (e. g., in Chapter 5) that CIDSs need to evolve from simple systems that manage alerts to full-featured systems capable of creating distributed intrusion detection models through cooperation. Advancements in distributed ML, we argue, are directly related to the future of CIDSs. Ensemble learning is a successful technique that intrinsically supports the development of distributed models. As researchers, we still need to interconnect more closely the fields of ensemble learning and CIDSs. The end goal should focus on developing CIDSs that operate at the *detection level*, and ensemble learning may provide the means to achieve this goal.

In the process of interconnecting ensemble learning and CIDSs together, the local detection and global detection components of the CIDS architecture need to be tied more closely together. As such, the components need to be addressed not as individual and independent components but, rather, as coupled components. For the global detection component to create intrusion detection models, the local detection component needs to extract network features or build models that align with the specific requirements of the global detection component.

We expect that the CIDS architecture experiences and evolves in a similar fashion to the OSI model. When the OSI model was originally conceptualized, each of its seven layers had well defined functionalities that did not overlap much with each other. At present, the OSI model is just but a theoretical concept used to understand the different components that enable network communication. Implementations of the OSI model dilute the functionality of the layers such that it is not possible to distinguish where the responsibilities of one layer start and end. Similarly, the CIDS architecture has components,

or layers, whose implementation can easily become tangled. There are a few places where this is most obvious. The *data correlation and aggregation* and the *data dissemination* components are most effectively implemented in combination. If we further wish to consider underlays in the dissemination process, then the *membership management* component should also be taken into account. The same goes for the *location detection* component and the *global detection* component.

We envision a treatment of the *membership management* component similarly to how we treated the *data dissemination* component. Instead of using existing protocols to manage members, we believe that CIDSs would benefit from the development of especially tailored membership management mechanisms. Such mechanism would need to take into account data correlation and aggregation strategies, and data dissemination methodologies to better optimize how members are organized.

BIBLIOGRAPHY

Abt, Sebastian and Harald Baier (2013). "Are We Missing Labels? A Study of the Availability of Ground-Truth in Network Security Research." In: *Badgers 2014* (cit. on pp. 42, 43, 46, 49, 187).

Akhtar, Naveed and Ajmal Mian (2018). "Threat of Adversarial Attacks on Deep Learning in Computer Vision: A Survey." In: *IEEE Access* 6, pp. 14410–14430 (cit. on p. 10).

Anceaume, Emmanuelle and Yann Busnel (2013). "Sketch *-metric: Comparing Data Streams via Sketching." In: *2013 IEEE 12th International Symposium on Network Computing and Applications*. IEEE, pp. 25–32 (cit. on p. 135).

Androutsellis-Theotokis, Stephanos, Diomidis Spinellis, Campbell Gibson, and Kay Jung (2004). "A survey of peer-to-peer content distribution technologies." In: *ACM computing surveys (CSUR)* 36.4, pp. 335–371 (cit. on p. 31).

Anwar, Shahid et al. (2017). "From intrusion detection to an intrusion response system: Fundamentals, requirements, and future directions." In: *Algorithms* 10.2 (cit. on p. 22).

Axelsson, Stefan (1998). *Research in intrusion-detection systems: A survey*. Tech. rep. Department of Computer Engineering, Chalmers University of Technology, p. 93 (cit. on p. 5).

Barbosa, Rafael Ramos Regis, Ramin Sadre, Aiko Pras, and Remco van de Meent (2010). "Simpleweb/university of twente traffic traces data repository." In: *Centre for Telematics and Information Technology University of Twente, Enschede, Technical Report* (cit. on p. 44).

Bärwolff, Matthias (2010). *End-to-End Arguments in the Internet: Principles, Practices, and Theory* (cit. on p. 1).

Bellovin, Steven M (1992). "Packets Found on an Internet 1 Introduction 2 Address Space Oddities." In: *Computer Communications* 23.3, pp. 1–8 (cit. on p. 51).

Bhuyan, Monowar H, Dhruba K Bhattacharyya, and Jugal K Kalita (2015). "Towards generating real-life datasets for network intrusion detection." In: *International Journal of Network Security* 17.6, pp. 683–701 (cit. on p. 40).

Bishop, Christopher M (2006). *Pattern recognition and machine learning*. springer (cit. on pp. 16, 163).

Boggs, Nathaniel, Sharath Hiremagalore, Angelos Stavrou, and Salvatore J. Stolfo (2011). "Cross-domain collaborative anomaly detection: So far yet so close." In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 6961 LNCS, pp. 142–160 (cit. on p. 138).

Bolstad, William M (2004). *Introduction to Bayesian Statistics*. John Wiley & Sons, Inc (cit. on p. 164).

Brauckhoff, Daniela and a Wagner (2008). "FLAME: a flow-level anomaly modeling engine." In: *The conference on Cyber security* (cit. on pp. 41, 47).

Breve, Fabricio A, Liang Zhao, and Marcos G Quiles (2010). "Semi-Supervised Learning from Imperfect Data through Particle Cooperation and Competition." In: *Neural Networks (IJCNN), The 2010 International Joint Conference on*, pp. 1–8 (cit. on p. 89).

Buchegg, S and J Y L Boudec (2004). "A Robust Reputation System for Peer-to-Peer and Mobile Ad-hoc Networks." In: *Proc. of the 2nd Workshop on the Economics of Peer-to-Peer Systems* (cit. on p. 164).

Burnett, Chris, Timothy J Norman, and Katia Sycara (2010). "Bootstrapping Trust Evaluations through Stereotypes." In: *9th International Conference on Autonomous Agents and Multiagent Systems*, pp. 241–248 (cit. on p. 180).

Butun, Ismail, Salvatore D. SD Morgera, and Ravi Sankar (2004). "A Survey of Intrusion Detection Systems in Wireless Sensor Networks." In: *Communications Surveys & Tutorials, IEEE* PP.99, pp. 1–17 (cit. on pp. 2, 21).

Bye, Rainer, Seyit Ahmet Camtepe, and Sahin Albayrak (2010). "Collaborative Intrusion Detection Framework: Characteristics, Adversarial Opportunities and Countermeasures." In: *CollSec* (cit. on p. 32).

Cai, Min, Kai Hwang, Yu-Kwong Kwok, Shanshan Song, and Yu Chen (2005). "Collaborative internet worm containment." In: *IEEE Security & Privacy* 3.3, pp. 25–33 (cit. on p. 109).

CAIDA (2018). *Center for Applied Internet Data Analysis*. URL: https://www.caida.org/home/ (visited on 12/06/2018) (cit. on pp. 5, 44).

Caragea, Doina, Adrian Silvescu, and Vasant Honavar (2004). "A Framework for Learning from Distributed Data Using Sufficient Statistics and its Application to Learning Decision Trees." In: *International Journal of Hybrid Intelligent Systems* 1.1, pp. 80–89 (cit. on p. 32).

Catania, Carlos a. and Carlos García Garino (2012). "Automatic network intrusion detection: Current techniques and open issues." In: *Computers & Electrical Engineering* 38.5, pp. 1062–1072 (cit. on p. 5).

Cha, Sung-hyuk (2007). "Comprehensive Survey on Distance / Similarity Measures between Probability Density Functions." In: *International Journal of Mathematical Models and Methods in Applied Sciences* 1.4, pp. 300–307 (cit. on pp. 133, 135).

Chan, Philip K and Salvatore J Stolfo (1993). "Toward Parallel and Distributed Learning by Meta-Learning." In: *AAAI workshop in Knowledge Discovery in Databases*, pp. 227–240 (cit. on p. 28).

Chandola, Varun, Arindam Banerjee, and Vipin Kumar (2009). "Anomaly detection: A Survey." In: *ACM Computing Surveys* 41.3, pp. 1–58 (cit. on pp. 20, 112).

Chen, Zhaomin, Chai Kiat Yeo, Bu Sung Lee, and Chiew Tong Lau (2016). "Detection of network anomalies using Improved-MSPCA with sketches." In: *Computers & Security* (cit. on pp. 24, 93).

Cho, Kenjiro, K Mitsuya, and a Kato (2000). "Traffic data repository at the WIDE project." In: *ATEC '00 Proceedings of the annual conference on USENIX Annual Technical Conference*, pp. 51–52 (cit. on pp. 89, 154, 156).

Cordero, Carlos Garcia, Sascha Hauke, Max Muhlhauser, and Mathias Fischer (2016). "Analyzing flow-based anomaly intrusion detection using Replicator Neural Networks." In: *2016 14th Annual Conference on Privacy, Security and Trust (PST)*. IEEE, pp. 317–324 (cit. on pp. 10, 75).

Cordero, Carlos Garcia, Giulia Traverso, et al. (2018). "Sphinx: a Colluder-Resistant Trust Mechanism for Collaborative Intrusion Detection." In: *IEEE Access*, pp. 1–13 (cit. on p. 10).

Cordero, Carlos Garcia, Emmanouil Vasilomanolakis, Nikolay Milanov, Christian Koch, David Hausheer, and Max Muhlhauser (2015). "ID2T: A DIY dataset creation toolkit for Intrusion Detection Systems." In: *2015 IEEE Conference on Communications and Network Security (CNS)*. IEEE, pp. 739–740 (cit. on pp. 9, 40, 48).

Cordero, Carlos Garcia, Emmanouil Vasilomanolakis, Max Mühlhäuser, and Mathias Fischer (2015). "Community-Based Collaborative Intrusion Detection." In: *International Conference on Security and Privacy in Communication Systems*, pp. 665–681 (cit. on p. 10).

Cormode, Graham and Muthu Muthukrishnan (2012). "Approximating Data with the Count-Min Sketch." In: *IEEE Software* 29.1, pp. 64–69 (cit. on p. 135).

Cormode, Graham and S. Muthukrishnan (2005). "An improved data stream summary: The count-min sketch and its applications." In: *Journal of Algorithms* 55.1, pp. 58–75 (cit. on pp. 134, 135).

Cotton, Michelle, Lars Eggert, Joe Touch, Magnus Westerlund, and Stuart Cheshire (2011). *Internet assigned numbers authority (IANA) procedures for the management of the service name and transport protocol port number registry*. Tech. rep. (cit. on p. 57).

Creech, Gideon and Jiankun Hu (2013). "Generation of a new IDS test dataset: Time to retire the KDD collection." In: *Wireless Communications and Networking ...* pp. 1–6 (cit. on pp. 35, 43).

Cretu, Gabriela F., Angelos Stavrou, Michael E. Locasto, Salvatore J. Stolfo, and Angelos D. Keromytis (2008). "Casting out demons: Sanitizing training data for anomaly sensors." In: *Proceedings - IEEE Symposium on Security and Privacy*, pp. 81–95 (cit. on p. 89).

Cuppens, Frédéric and Alexandre Miège (2002). "Alert correlation in a cooperative intrusion detection framework." In: *IEEE Symposium on Security and Privacy (S&P)*. IEEE (cit. on p. 27).

Danzig, Peter B. and Sugih Jamin (1991). "tcplib: A Library of Internetwork Traffic Characteristics." In: *Library* 48, pp. 1–8 (cit. on p. 52).

Das, Kaustav and Jeff Schneider (2007). "Detecting anomalous records in categorical datasets." In: *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '07*, p. 220 (cit. on p. 21).

Dash, Denver, B Kveton, and JM Agosta (2006). "When gossip is good: Distributed probabilistic inference for detection of slow network intrusions." In: *Proceedings of the ...* pp. 1115–1122 (cit. on p. 28).

Dau, Hoang Anh, Vic Ciesielski, and Andy Song (2014). "Anomaly Detection Using Replicator Neural Networks Trained on Examples of One Class." In: *Simulated Evolution and Learning*. Dunedin, New Zealand: Springer International Publishing, pp. 311–322 (cit. on pp. 24, 91, 103).

Daubert, Jörg, Mathias Fischer, Tim Grube, Stefan Schiffner, Panayotis Kikiras, and Max Mühlhäuser (2016). "Anonpubsub: Anonymous publish-subscribe overlays." In: *Computer Communications* 76, pp. 42–53 (cit. on p. 32).

Deng, Li and Dong Yu (2014). "Deep Learning: Methods and Applications." In: *Foundations and Trends in Signal Processing* 7.3-4, pp. 197–387 (cit. on pp. 87, 104).

Dizaji, Kamran Ghasedi, Amirhossein Herandi, Cheng Deng, Weidong Cai, and Heng Huang (2017). "Deep clustering via joint

convolutional autoencoder embedding and relative entropy minimization." In: *Computer Vision (ICCV), 2017 IEEE International Conference on*. IEEE, pp. 5747–5756 (cit. on p. 18).

Duchi, John, Elad Hazan, and Yoram Singer (2011). "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization." In: *Jmlr* 12, pp. 1–40 (cit. on p. 91).

Duda, Richard O, Peter E Hart, and David G Stork (2012). *Pattern classification*. John Wiley & Sons (cit. on p. 133).

Duma, Claudiu, Martin Karresand, Nahid Shahmehri, and Germano Caronni (2006). "A trust-aware, P2P-based overlay for intrusion detection." In: *Proceedings - International Workshop on Database and Expert Systems Applications, DEXA*, pp. 692–697 (cit. on p. 162).

Dumoulin, Vincent and Francesco Visin (2016). "A guide to convolution arithmetic for deep learning." In: pp. 1–28 (cit. on p. 104).

Dwarakanath, Rahul, Boris Koldehofe, Yashas Bharadwaj, The An Binh Nguyen, David Eyers, and Ralf Steinmetz (2017). "TrustCEP: Adopting a trust-based approach for distributed complex event processing." In: *Proceedings - 18th IEEE International Conference on Mobile Data Management, MDM 2017* May, pp. 30–39 (cit. on p. 165).

Eckmann, Steven T, Giovanni Vigna, and Richard A Kemmerer (2002). "STATL: An attack language for state-based intrusion detection." In: *Journal of computer security* 10.1-2, pp. 71–103 (cit. on p. 32).

Estevez-Tapiador, Juan M., Pedro Garcia-Teodoro, and Jesus E. Diaz-Verdejo (2004). "Anomaly detection methods in wired networks: a survey and taxonomy." In: *Computer Communications* 27.16, pp. 1569–1584 (cit. on p. 24).

Fan, Li, Pei Cao, Jussara Almeida, and Andrei Z. Broder (2000). "Summary cache: A scalable wide-area Web cache sharing protocol." In: *IEEE/ACM Transactions on Networking* 8.3, pp. 281–293 (cit. on p. 134).

Folino, Gianluigi and Pietro Sabatino (2016). "Ensemble based collaborative and distributed intrusion detection systems: A survey." In: *Journal of Network and Computer Applications* 66, pp. 1–16 (cit. on p. 112).

Fontugne, Romain, Pierre Borgnat, Patrice Abry, and Kensuke Fukuda (2010). "MAWILab: combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking." In: *Proceedings of the 6th International Conference (Co-NEXT '10)*. New York, New York, USA: ACM Press, 8:1–8:12 (cit. on pp. 5, 43, 89).

Fung, Carol J., Olga Baysal, Jie Zhang, Issam Aib, and Raouf Boutaba (2008). "Trust management for host-based collaborative intrusion detection." In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 5273 LNCS, pp. 109–122 (cit. on p. 165).

Fung, Carol J., Jie Zhang, Issam Aib, and Raouf Boutaba (2009). "Robust and scalable trust management for collaborative intrusion detection." In: *2009 IFIP/IEEE International Symposium on Integrated Network Management, IM 2009*, pp. 33–40 (cit. on p. 165).

Fung, Carol J., Jie Zhang, Issam Aib, and Raouf Boutaba (2011). "Dirichlet-based trust management for effective collaborative intrusion detection networks." In: *IEEE Transactions on Network and Service Management* 8.2, pp. 79–91 (cit. on pp. 6, 161, 165).

Fung, Carol, Jie Zhang, Issam Aib, and Raouf Boutaba (2011). "Trust management and admission control for host-based collaborative intrusion detection." In: *Journal of Network and Systems Management* 19.2, pp. 257–277 (cit. on p. 166).

Gamer, Thomas (2012). "Collaborative anomaly-based detection of large-scale internet attacks." In: *Computer Networks* 56.1, pp. 169–185 (cit. on pp. 28, 109, 137).

Garcia, Joaquin, Fabien Autrel, Joan Borrell, Sergio Castillo, Frederic Cuppens, and Guillermo Navarro (2004a). "Decentralized Publish-Subscribe System to Prevent Coordinated Attacks via Alert Correlation." In: *Information and Communications Security* 3269.October 2014, pp. 297–304 (cit. on p. 28).

Garcia, Joaquin, Fabien Autrel, Joan Borrell, Sergio Castillo, Frederic Cuppens, and Guillermo Navarro (2004b). "Decentralized publish-subscribe system to prevent coordinated attacks via alert correlation." In: *International Conference on Information and Communications Security*. Springer, pp. 223–235 (cit. on p. 32).

Gazis, Vangelis, Carlos Garcia Cordero, Emmanouil Vasilomanolakis, Panayotis Kikiras, and Alex Wiesmaier (2014). "Security perspectives for collaborative data acquisition in the internet of things." In: *International Internet of Things Summit*. Springer, pp. 271–282 (cit. on p. 10).

Gil Pérez, Manuel, Félix Gómez Mármol, Gregorio Martínez Pérez, and Antonio F. Skarmeta Gómez (2013). "RepCIDN: A reputation-based collaborative intrusion detection network to lessen the impact of malicious alarms." In: *Journal of Network and Systems Management* 21.1, pp. 128–167 (cit. on pp. 165, 166).

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. MIT Press (cit. on p. 19).

Grossman, Dan (2002). *New terminology and clarifications for diffserv*. RFC 3260 (cit. on p. 51).

Gupta, I., A.M. Kermarrec, and A.J. Ganesh (2010). "Efficient epidemic-style protocols for reliable and scalable multicast." In: *21st IEEE Symposium on Reliable Distributed Systems, 2002. Proceedings.* 3. IEEE Comput. Soc, pp. 180–189 (cit. on p. 6).

Haider, W., J. Hu, J. Slay, B. P. Turnbull, and Y. Xie (2017). "Generating realistic intrusion detection system dataset based on fuzzy qualitative modeling." In: *Journal of Network and Computer Applications* 87.November 2016, pp. 185–192 (cit. on p. 46).

Hang, Chung-Wei and Munindar P Singh (2011). "Trustworthy Service Selection and Composition." In: *ACM Trans. Auton. Adapt. Syst.* 6.1, 5:1–5:17 (cit. on p. 164).

Hartono, Pitoyo and Shuji Hashimoto (2007). "Learning from imperfect data." In: *Applied Soft Computing Journal* 7.1, pp. 353–363 (cit. on p. 89).

Hawkins, Simon, Hongxing He, Graham Williams, and Rohan Baxter (2002). "Outlier detection using replicator neural networks." In: *Data Warehousing and ...* (Cit. on pp. 87, 103).

Hecht-Nielsen, R (1995). "Replicator neural networks for universal optimal source coding." In: *Science (New York, N.Y.)* 269.5232, pp. 1860–1863 (cit. on pp. 87, 89).

Hong, Junho and Chen-Ching Liu (2017). "Intelligent electronic devices with collaborative intrusion detection systems." In: *IEEE Transactions on Smart Grid* (cit. on p. 28).

Huang, Jingwei, Zbigniew Kalbarczyk, and David M Nicol (2014). "Knowledge discovery from big data for intrusion detection using LDA." In: *2014 IEEE International Congress on Big Data*. IEEE, pp. 760–761 (cit. on p. 113).

IMPACT (2017). *Information Marketplace*. URL: https://www.impactcybertrust.org (visited on 08/17/2018) (cit. on p. 45).

Inacio, Christopher M. and Brian Trammell (2010). "Yet Another Flowmeter (YAF)." In: *International conference on large installation system administration (LISA)* (cit. on pp. 1, 95).

Jiang, Dingde, Zhengzheng Xu, Peng Zhang, and Ting Zhu (2014). "A transform domain-based anomaly detection approach to network-wide traffic." In: *Journal of Network and Computer Applications* 40.1, pp. 292–306 (cit. on p. 24).

Josang, Audun and Roslan Ismail (2002). "The beta reputation system." In: *Proceedings of the 15th bled electronic commerce conference*. Vol. 5, pp. 2502–2511 (cit. on p. 164).

Kanda, Yoshiki, Romain Fontugne, Kensuke Fukuda, and Toshiharu Sugawara (2013). "ADMIRE: Anomaly detection method using entropy-based PCA with three-step sketches." In: *Computer Communications* 36.5, pp. 575–588 (cit. on pp. 93, 94).

Kannadiga, Pradeep and Mohammad Zulkernine (2005). "DIDMA : A Distributed Intrusion Detection System Using Mobile Agents." In: *International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*. IEEE, pp. 238–245 (cit. on p. 27).

Karlik, Bekir and A Vehbi Olgac (2011). "Performance analysis of various activation functions in generalized MLP architectures of neural networks." In: *International Journal of Artificial Intelligence and Expert Systems* 1.4, pp. 111–122 (cit. on pp. 87, 92).

KDD Cup 99 (1999). *Knowledge Discovery and Data Mining Tools Competition*. URL: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html (visited on 08/20/2008) (cit. on p. 43).

Kermarrec, Anne-Marie and Maarten van Steen (2007). "Gossiping in distributed systems." In: *SIGOPS Oper. Syst. Rev.* 41.5, pp. 2–7 (cit. on pp. 31, 137, 188).

Kim, Hyang-Ah and Brad Karp (2004). "Autograph: Toward Automated, Distributed Worm Signature Detection." In: *USENIX security symposium*. Vol. 286. San Diego, CA (cit. on p. 23).

Koch, Robert, Mario Golling, and Gabi Dreo Rodosek (2014). "Towards Comparability of Intrusion Detection Systems: New Data Sets." In: *TERENA Networking Conference*, p. 7 (cit. on pp. 37, 40).

Koller, Daphne and Nir Friedman (2009). *Probabilistic graphical models: principles and techniques*. MIT press (cit. on pp. 144, 145, 149, 155).

Kostić, Dejan, Adolfo Rodriguez, Jeannie Albrecht, and Amin Vahdat (2003). "Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh." In: *SOSP '03 Proceedings of the nineteenth ACM symposium on Operating systems principles* 37.5, pp. 282–297 (cit. on pp. 32, 134).

Kreibich, Christian and Jon Crowcroft (2004). "Honeycomb: creating intrusion detection signatures using honeypots." In: *ACM SIGCOMM computer communication review* 34.1, pp. 51–56 (cit. on p. 23).

Kwon, Donghwoon, Hyunjoo Kim, Jinoh Kim, Sang C. Suh, Ikkyun Kim, and Kuinam J. Kim (2017). "A survey of deep learning-based network anomaly detection." In: *Cluster Computing* (cit. on pp. 21, 22).

Lakhina, Anukool, Mark Crovella, and Christiphe Diot (2004). "Characterization of network-wide anomalies in traffic flows." In: *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement - IMC '04* 6, p. 201 (cit. on pp. 25, 83).

Lakhina, Anukool, Mark Crovella, and Christophe Diot (2004). "Diagnosing network-wide traffic anomalies." In: *ACM SIGCOMM Computer Communication Review* 34, p. 219 (cit. on pp. 22, 24, 85, 94).

Lakhina, Anukool, Mark Crovella, and Christophe Diot (2005). "Mining anomalies using traffic feature distributions." In: *Conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM)05*. New York, New York, USA: ACM Press, pp. 217–228 (cit. on pp. 86, 89, 101).

Lassoued, Imed (2011). "Adaptive Monitoring and Management of Internet Traffic." PhD Thesis. Université de Nice, p. 160 (cit. on p. 61).

Lauritzen, Steffen L (1996). *Graphical models*. Vol. 17. Clarendon Press (cit. on p. 164).

Lavalle, Steve, Eric Lesser, Rebecca Shockley, Michael S Hopkins, and Nina Kruschwitz (2011). "Big data, analytics and the path from insights to value." In: *MIT sloan management review* 52.2, p. 21 (cit. on p. 113).

Lazarevic, Aleksandar, Vipin Kumar, and Jaideep Srivastava (2005). "Intrusion detection: A survey." In: *Managing Cyber Threats*. Springer, pp. 19–78 (cit. on p. 2).

Lazarevic, Aleksandar, Nisheeth Srivastava, Ashutosh Tiwari, Josh Isom, Nikunj Oza, and Jaideep Srivastava (2009). "Theoretically Optimal Distributed Anomaly Detection." In: *2009 IEEE International Conference on Data Mining Workshops*, pp. 515–520 (cit. on pp. 28, 102, 131, 133).

Lee, Wenke, Wei Fan, Matthew Miller, Salvatore J. Stolfo, and Erez Zadok (2002). "Toward cost-sensitive modeling for intrusion detection and response." In: *Journal of Computer Security* 10.1-2, pp. 5–22 (cit. on p. 22).

Li, Bingdong, Jeff Springer, George Bebis, and Mehmet Hadi Gunes (2013). "A survey of network flow applications." In: *Journal of Network and Computer Applications* 36.2, pp. 567–581 (cit. on pp. 22, 89).

Li, Xin et al. (2006). "Detection and identification of network anomalies using sketch subspaces." In: *Proceedings of the 6th ACM SIGCOMM on Internet measurement - IMC '06*, p. 147 (cit. on pp. 24, 86, 93, 94).

Li, Zhichun, Yan Chen, and Aaron Beach (2006). "Towards scalable and robust distributed intrusion alert fusion with good load balancing." In: *SIGCOMM workshop on Large-scale attack defense (LSAD)*. New York, New York, USA: ACM Press, pp. 115–122 (cit. on p. 27).

Liberatore, Marc and Prashant Shenoy (2018). *Umass trace repository*. URL: http://traces.cs.umass.edu (visited on 08/12/2018) (cit. on p. 45).

Lin, Jianhua (1991). "Divergence Measures Based on the Shannon Entropy." In: *IEEE Transactions on Information Theory* 37.1, pp. 145–151 (cit. on pp. 139, 141).

Lippmann, R.P. et al. (1999). "Evaluating intrusion detection systems without attacking your friends: The 1998 DARPA intrusion detection evaluation." In: *DARPA Information Survivability Conference and Exposition, 2000. DISCEX '00. Proceedings*, 12–26 vol.2 (cit. on pp. 5, 37, 43).

Liu, Guisong, Zhang Yi, and Shangming Yang (2007). "A hierarchical intrusion detection model based on the PCA neural networks." In: *Neurocomputing* 70.7-9, pp. 1561–1568 (cit. on p. 89).

Liu, Qiang, Pan Li, Wentao Zhao, Wei Cai, Shui Yu, and Victor C M Leung (2018). "A survey on security threats and defensive techniques of machine learning: a data driven view." In: *IEEE access* 6, pp. 12103–12117 (cit. on p. 10).

Liu, Xiaoxue, Peidong Zhu, Yan Zhang, and Kan Chen (2015). "A collaborative intrusion detection mechanism against false data injection attack in advanced metering infrastructure." In: *IEEE Transactions on Smart Grid* 6.5, pp. 2435–2443 (cit. on p. 28).

Liu, Xin, Anwitaman Datta, Krzysztof Rzadca, and Ee-Peng Lim (2009). "StereoTrust: A Group Based Personalized Trust Model." In: *Proceedings of the 18th ACM Conference on Information and Knowledge Management*. CIKM '09. New York, NY, USA: ACM, pp. 7–16 (cit. on p. 181).

Liu, Xin, Gilles Tredan, and Anwitaman Datta (2014). "A Generic Trust Framework for Large-Scale Open Systems Using Machine Learning." In: *Computational Intelligence* 30.4, pp. 700–721 (cit. on p. 165).

Liu, Zhifa, Brandon Malone, and Changhe Yuan (2012). "Empirical evaluation of scoring functions for Bayesian network model selection." In: *BMC bioinformatics* 13.15, S14 (cit. on p. 144).

Locasto, M, J Parekh, A Keromytis, and S; Stolfo (2005). "Towards Collaborative Security and P2P Intrusion Detection." In: *IEEE*

Workshop on Information Assurance and Security. IEEE, pp. 333–339 (cit. on pp. 32, 133).

Lu, Wei and Issa Traore (2005). "A new unsupervised anomaly detection framework for detecting network attacks in real-time." In: *Cryptology and Network Security*, pp. 96–109 (cit. on p. 89).

Maclin, Richard and David Opitz (2011). "Popular ensemble methods: An empirical study." In: *CoRR* abs/1106.0, pp. 169–198 (cit. on p. 112).

Mahoney V., Matthew and Philip Chan K. (2001). *PHAD: packet header anomaly detection for identifying hostile network traffic* (cit. on p. 112).

Mahoney, Matthew V. and Philip K. Chan (2002). "Learning nonstationary models of normal network traffic for detecting novel attacks." In: *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '02*, p. 376 (cit. on pp. 109, 112).

Mahoney, Matthew V and P.K. Chan (2003). "Learning rules for anomaly detection of hostile network traffic." In: *Third IEEE International Conference on Data Mining*, pp. 601–604 (cit. on pp. 24, 40, 110, 111, 113, 122).

Mahoney, Matthew V and Philip K Chan (2003). "An Analysis of the 1999 DARPA/Lincoln Laboratory Evaluation Data for Network Anomaly Detection." In: *In Proceedings of the Sixth International Symposium on Recent Advances in Intrusion Detection* 2820.Ll, pp. 220–237 (cit. on pp. 35, 37, 43, 51, 60).

Marchetti, Mirco, Michele Messori, and Michele Colajanni (2009). "Peer-to-Peer Architecture for Collaborative Intrusion and Malware Detection on a Large Scale." In: *Lecture Notes in Computer Science* 5735, pp. 475–490 (cit. on pp. 27, 28, 30).

Markham, T. and C. Payne (2001). "Security at the network edge: a distributed firewall architecture." In: *Proceedings DARPA Information Survivability Conference and Exposition II. DISCEX'01*. Vol. 1. IEEE Comput. Soc, pp. 279–286 (cit. on p. 2).

McFowland, E, S Speakman, and DB Neill (2013). "Fast generalized subset scan for anomalous pattern detection." In: *The Journal of Machine Learning ...* 14, pp. 1533–1561 (cit. on pp. 20, 21).

McHugh, John (2000). "Testing Intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory." In: *ACM Transactions on Information and System Security* 3.4, pp. 262–294 (cit. on p. 51).

Mee, Paul and Rico Brandenburg (2018). *Large-Scale Cyber-Attacks On The Financial System*. URL: https://www.oliverwyman.com/our-

expertise/insights/2018/mar/large-scale-cyber-attacks-on-the-financial-system.html (visited on 12/04/2018) (cit. on p. 2).

Mikolov, Tomas, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean (2013). "Distributed representations of words and phrases and their compositionality." In: *Advances in neural information processing systems*, pp. 3111–3119 (cit. on p. 18).

Mitchell, Robert and Ing-Ray Chen (2014). "A survey of intrusion detection techniques for cyber-physical systems." In: *ACM Computing Surveys (CSUR)* 46.4, p. 55 (cit. on p. 2).

Moustafa, Nour and Jill Slay (2015). "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)." In: *2015 Military Communications and Information Systems Conference (MilCIS)* December, pp. 1–6 (cit. on p. 45).

Nechaev, Boris, Mark Allman, Vern Paxson, and Andrei V Gurtov (2010). "A Preliminary Analysis of TCP Performance in an Enterprise Network." In: *INM/WREN* 10 (cit. on p. 44).

Nesterov, Yurii (1983). "A method for unconstrained convex minimization problem with the rate of convergence o(1/k^2)." In: *Doklady AN USSR* 269, pp. 543–547 (cit. on pp. 91, 96).

Neustar (2014). *Distribution of Estimated Average Costs per Hour of Ddos Attacks in The United Kingdom (Uk) in 2012 and 2013*. URL: www.statista.com/statistics/463505/estimated-average-costs-per-hour-of-ddos-attacks-in-the-united-kingdom-uk/ (visited on 12/05/2018) (cit. on p. 2).

Nielsen, Mogens, Karl Krukow, and Vladimiro Sassone (2007). "A bayesian model for event-based trust." In: *Electronic Notes in Theoretical Computer Science* 172, pp. 499–521 (cit. on p. 164).

Nychis, George, Vyas Sekar, David G. Andersen, Hyong Kim, and Hui Zhang (2008). "An empirical evaluation of entropy-based traffic anomaly detection." In: *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement conference - IMC '08*. New York, New York, USA: ACM Press, p. 151 (cit. on pp. 25, 89, 133).

Oikonomou, George, Jelena Mirkovic, Peter Reiher, and Max Robinson (2006). "A framework for a collaborative DDoS defense." In: *Proceedings - Annual Computer Security Applications Conference, ACSAC*, pp. 33–42 (cit. on p. 137).

Oppliger, Rolf (2001). *Internet and intranet security*. Artech House (cit. on p. 1).

Paxson, Vern (1999). "Bro: A system for detecting network intruders in real-time." In: *Computer Networks* 31.23, pp. 2435–2463 (cit. on pp. 23, 30, 51, 76).

Peteiro-Barral, Diego and Bertha Guijarro-Berdiñas (2013). "A survey of methods for distributed machine learning." In: *Progress in Artificial Intelligence* 2.1, pp. 1–11 (cit. on pp. 28, 113).

Pimentel, Marco a F, David a. Clifton, Lei Clifton, and Lionel Tarassenko (2014). "A review of novelty detection." In: *Signal Processing* 99, pp. 215–249 (cit. on p. 24).

Postel, Jon et al. (1981). *Internet protocol*. RFC 791 (cit. on p. 51).

PROTECTIVE: Proactive Risk Management (2018). URL: https://protective-h2020.eu/background/ (visited on 12/14/2018) (cit. on p. 187).

Quittek, J., T. Zseby, B. Claise, and S. Zander (2004). "Requirements for IP Flow Information Export (IPFIX)." In: *IETF RFC 3917*, pp. 1–34 (cit. on p. 85).

Ramakrishnan, K K, S Floyd, and S Black (2001). *The addition of explicit congestion notification (ECN) to IP*. Tech. rep. (cit. on p. 58).

Reed, Erik and Ole J. Mengshoel (2014). "Bayesian network parameter learning using EM with parameter sharing." In: *CEUR Workshop Proceedings* 1218, pp. 48–59 (cit. on pp. 136, 145).

Reynolds, Joyce and Jon Postel (1994). *Assigned numbers*. Tech. rep. (cit. on p. 57).

Ries, Sebastian (2009). "Extending Bayesian trust models regarding context-dependence and user friendly representation." In: *Proceedings of the 2009 {ACM} Symposium on Applied Computing (SAC), Honolulu, Hawaii, USA*, pp. 1294–1301 (cit. on p. 164).

Ringberg, Haakon, Augustin Soule, Jennifer Rexford, and Christophe Diot (2007). "Sensitivity of PCA for traffic anomaly detection." In: *ACM SIGMETRICS Performance Evaluation Review* 35.1, p. 109 (cit. on pp. 24, 25).

Rodriguez, Miguel, Diego M Escalante, and Antonio Peregrin (2011). "Efficient distributed genetic algorithm for rule extraction." In: *Applied soft computing* 11.1, pp. 733–743 (cit. on p. 113).

Roesch, Martin (1999). "Snort: Lightweight intrusion detection for networks." In: *Proceedings of LISA '99: 13th Systems Administration Conference*. Vol. 99. 1. USENIX Association, pp. 229–238 (cit. on pp. 23, 30).

Roweis, Sam T and Lawrence K Saul (2000). "Nonlinear dimensionality reduction by locally linear embedding." In: *science* 290.5500, pp. 2323–2326 (cit. on p. 86).

Sallay, Hassen, Adel Ammar, Majdi Ben Saad, and Sami Bourouis (2013). "A Real Time Adaptive Intrusion Detection Alert Classifier for High Speed Networks." In: *2013 IEEE 12th International Symposium on Network Computing and Applications*, pp. 73–80 (cit. on p. 89).

Sangster, Benjamin et al. (2009). "Toward Instrumenting Network Warfare Competitions to Generate Labeled Datasets." In: *CSET* (cit. on p. 45).

Sedjelmaci, Hichem and Sidi Mohammed Senouci (2015). "An accurate and efficient collaborative intrusion detection framework to secure vehicular networks." In: *Computers & Electrical Engineering* 43, pp. 33–47 (cit. on p. 28).

Shiravi, Ali, Hadi Shiravi, Mahbod Tavallaee, and Ali a. Ghorbani (2012). "Toward developing a systematic approach to generate benchmark datasets for intrusion detection." In: *Computers & Security* 31.3, pp. 357–374 (cit. on pp. 40, 41, 48, 52).

Sommer, Robin and Vern Paxson (2010). "Outside the Closed World: On Using Machine Learning for Network Intrusion Detection." In: *2010 IEEE Symposium on Security and Privacy*, pp. 305–316 (cit. on p. 22).

Song, Jungsuk, Hiroki Takakura, and Yasuo Okabe (2006). "Description of kyoto university benchmark data." In: *Academic Center for Computing and Media Studies (ACCMS), Kyoto University* (cit. on p. 44).

Song, Jungsuk, Hiroki Takakura, and Yasuo Okabe (2008). "Cooperation of intelligent honeypots to detect unknown malicious codes." In: *Information Security Threats Data Collection and Sharing, 2008. WISTDCS'08. WOMBAT Workshop on*. IEEE, pp. 31–39 (cit. on p. 44).

Song, Xiuyao, Mingxi Wu, Christopher Jermaine, and Sanjay Ranka (2007). "Conditional anomaly detection." In: *IEEE Transactions on Knowledge and Data Engineering* 19.5, pp. 631–645 (cit. on p. 20).

Soysal, Murat and Ece Guran Schmidt (2010). "Machine learning algorithms for accurate flow-based network traffic classification: Evaluation and comparison." In: *Performance Evaluation* 67.6, pp. 451–467 (cit. on pp. 22, 88).

Sperotto, Anna and Aiko Pras (2010). "Flow-based intrusion detection." Ph.D. Thesis. University of Twente, p. 182 (cit. on pp. 3, 22).

Sperotto, Anna, Ramin Sadre, Frank Van Vliet, and Aiko Pras (2009). "A labeled data set for flow-based intrusion detection." In: *International Workshop on IP Operations and Management*. Springer, pp. 39–50 (cit. on p. 44).

Srivastava, Nitish, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov (2014). "Dropout : A Simple Way to Prevent Neural Networks from Overfitting." In: *Journal of Machine Learning Research (JMLR)* 15, pp. 1929–1958 (cit. on pp. 91, 92).

Sun, Pei, Sanjay Chawla, and Bavani Arunasalam (2006). "Mining for outliers in sequential databases." In: *Proceedings of the 2006 SIAM International Conference on Data Mining*. SIAM, pp. 94–105 (cit. on p. 21).

Sutskever, I and J Martens (2013). "On the importance of initialization and momentum in deep learning." In: *Proceedings of the 30th international conference on machine learning (ICML-13)*, pp. 1139–1147 (cit. on p. 91).

Tang, Jiliang, Huiji Gao, Huan Liu, and Atish Das Sarma (2012). "eTrust: Understanding Trust Evolution in an Online World." In: *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '12. New York, NY, USA: ACM, pp. 253–261 (cit. on p. 165).

Tavallaee, Mahbod, Ebrahim Bagheri, Wei Lu, and Ali A. Ghorbani (2009). "A detailed analysis of the KDD CUP 99 data set." In: *IEEE Symposium on Computational Intelligence for Security and Defense Applications, CISDA 2009* Cisda, pp. 1–6 (cit. on pp. 5, 45).

Tellenbach, Bernhard, Martin Burkhart, Dominik Schatzmann, and David Gugelmann (2011). "Accurate network anomaly classification with generalized entropy metrics." In: *Computer Networks* 55.15, pp. 3485–3502 (cit. on pp. 25, 86).

The Connected Consumer Survey (2017). URL: https://www.consumerbarometer.com (visited on 12/03/2018) (cit. on p. 1).

The state of the internet / security report (2018). URL: https://www.akamai.com/uk/en/multimedia/documents/case-study/spring-2018-state-of-the-internet-security-report.pdf (visited on 10/15/2018) (cit. on p. 77).

Timberg, Craig (2015). *Net of Insecurity: A Flaw in the Design*. URL: https://www.washingtonpost.com/sf/business/2015/05/30/net-of-insecurity-part-1 (visited on 12/02/2018) (cit. on p. 1).

Tóth, L and G Gosztolya (2004). "Replicator Neural Networks for Outlier Modeling in Segmental Speech Recognition." In: *Advances in Neural Networks–ISNN 2004*, pp. 996–1001 (cit. on pp. 92, 104).

Traverso, Giulia et al. (2017). "Evidence-Based Trust Mechanism Using Clustering Algorithms for Distributed Storage Systems." In: *15th Annual Conference on Privacy, Security and Trust (PST)* (cit. on p. 10).

Vasilomanolakis, Emmanouil, Carlos Garcia Cordero, Nikolay Milanov, and Max Muhlhauser (2016). "Towards the creation of synthetic, yet realistic, intrusion detection datasets." In: *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*. IEEE, pp. 1209–1214 (cit. on pp. 9, 40, 48).

Vasilomanolakis, Emmanouil, Shankar Karuppayah, Max Mühlhäuser, and Mathias Fischer (2015). "Taxonomy and Survey of Collaborative Intrusion Detection." In: *ACM Computing Surveys* 47.4, pp. 1–33 (cit. on pp. 4, 6, 26, 28, 29, 40, 44, 129, 131, 161, 165, 187).

Vasilomanolakis, Emmanouil, Matthias Krugl, Carlos Garcia Cordero, Max Muhlhauser, and Mathias Fischer (2016). "SkipMon: A locality-aware Collaborative Intrusion Detection System." In: *2015 IEEE 34th International Performance Computing and Communications Conference, IPCCC 2015* (cit. on pp. 3, 10, 40, 133, 134, 137).

Vasilomanolakis, Emmanouil, Shreyas Srinivasa, Carlos Garcia Cordero, and Max Muhlhauser (2016). "Multi-stage attack detection and signature generation with ICS honeypots." In: *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*. Vol. 2016. IEEE, pp. 1227–1232 (cit. on pp. 10, 23).

Vasilomanolakis, Emmanouil, Shreyas Srinivasa, and Max Mühlhäuser (2015). "Did you really hack a nuclear power plant? An industrial control mobile honeypot." In: *Communications and Network Security (CNS), 2015 IEEE Conference on*. IEEE, pp. 729–730 (cit. on p. 32).

Vasilomanolakis, Emmanouil, Michael Stahn, Carlos Garcia Cordero, and Max Muhlhauser (2015). "Probe-response attacks on collaborative intrusion detection systems: Effectiveness and countermeasures." In: *2015 IEEE Conference on Communications and Network Security (CNS)*. IEEE, pp. 699–700 (cit. on p. 10).

Vasilomanolakis, Emmanouil, Michael Stahn, Carlos Garcia Cordero, and Max Muhlhauser (2016). "On probe-response attacks in Collaborative Intrusion Detection Systems." In: *2016 IEEE Conference on Communications and Network Security (CNS)*. IEEE, pp. 279–286 (cit. on p. 10).

Verisign (2018). *Distributed Denial of Service Trends Report*. URL: https://www.verisign.com/assets/report-ddos-trends-Q12018%7B%5C_%7Den%7B%5C_%7DGB.pdf (visited on 12/05/2018) (cit. on p. 2).

Wagner, Kurt (2018). *Roughly one in four Americans is online 'constantly'*. URL: https://www.recode.net/2018/3/17/17130688/online-data-america-pew-research-obsession-facebook-google (visited on 12/03/2018) (cit. on p. 1).

Wang, Ke, Janak J. Parekh, and Salvatore J. Stolfo (2006). "Anagram: A Content Anomaly Detector Resistant to Mimicry Attack." In: pp. 226–248 (cit. on p. 138).

Wold, Svante, Kim Esbensen, and Paul Geladi (1987). "Principal component analysis." In: *Chemometrics and intelligent laboratory systems* 2.1-3, pp. 37–52 (cit. on p. 86).

Worldwide Infrastructure Security Report (2014). Tech. rep. Arbor Networks, p. 83 (cit. on pp. 1, 39).

Wu, Yu Sung, Bingrui Foo, Yongguo Mei, and Saurabh Bagchi (2003). "Collaborative intrusion detection system (CIDS): A framework for accurate and efficient IDS." In: *Proceedings - Annual Computer Security Applications Conference, ACSAC* 2003-Janua.Acsac, pp. 234–244 (cit. on p. 137).

Yan, Jeff and Pook Leong Cho (2006). "Enhancing collaborative spam detection with bloom filters." In: *Proceedings - Annual Computer Security Applications Conference, ACSAC*, pp. 414–425 (cit. on p. 137).

Yu, Jinqiao, Y V Ramana Reddy, Sentil Selliah, Sumitra Reddy, Vijayanand Bharadwaj, and Srinivas Kankanahalli (2005). "TRINETR: An architecture for collaborative intrusion detection and knowledge-based alert evaluation." In: *Advanced Engineering Informatics* 19.2, pp. 93–101 (cit. on p. 4).

Zhang, Bin, Jiahai Yang, Jianping Wu, Donghong Qin, and Lei Gao (2012). "PCA-subspace method - Is it good enough for network-wide anomaly detection." In: *Proceedings of the 2012 IEEE Network Operations and Management Symposium, NOMS 2012*, pp. 359–367 (cit. on pp. 18, 21).

Zhang, Qi and Ramaprabhu Janakiraman (2003). "Indra : A Distributed Approach to Network Intrusion Detection and Prevention." In: *Access* WUCS-01-30, pp. 1–6 (cit. on p. 28).

Zhang, Zheng, Jun Li, C N Manikopoulos, Jay Jorgenson, and Jose Ucles (2001). "HIDE : a Hierarchical Network Intrusion Detection System Using Statistical Preprocessing and Neural Network Classification." In: *IEEE Workshop on Information Assurance and Security*. IEEE, pp. 85–90 (cit. on p. 27).

Zhou, C V, S Karunasekera, and C Leckie (2007). "Evaluation of a Decentralized Architecture for Large Scale Collaborative Intrusion Detection." In: *IFIP/IEEE International Symposium on Integrated Network Management*. IEEE, pp. 80–89 (cit. on p. 27).

Zhou, Chenfeng Vincent, Christopher Leckie, and Shanika Karunasekera (2010a). "A survey of coordinated attacks and collaborative intrusion detection." In: *Computers & Security* 29.1, pp. 124–140 (cit. on p. 4).

Zhou, Chenfeng Vincent, Christopher Leckie, and Shanika Karunasekera (2010b). "A survey of coordinated attacks and collaborative intrusion detection." In: *Computers Security* 29.1, pp. 124–140 (cit. on p. 6).

Ziviani, Artur, Antonio Gomes, Marcelo Monsores, and Paulo Rodrigues (2007). "Network anomaly detection using nonextensive entropy." In: *IEEE Communications Letters* 11.12, pp. 1034–1036 (cit. on p. 25).

Zuech, Richard, Taghi M. Khoshgoftaar, Naeem Seliya, Maryam M. Najafabadi, and Clifford Kemp (2015a). "A New Intrusion Detection Benchmarking System." In: *Proceedings of the Twenty-Eighth International Florida Artificial Intelligence Research Society Conference* McHugh, pp. 252–255 (cit. on p. 45).

Zuech, Richard, Taghi M Khoshgoftaar, and Randall Wald (2015b). "Intrusion detection and big heterogeneous data: a survey." In: *Journal of Big Data* 2.1, p. 3 (cit. on p. 113).

Zulkernine, Mohammad and Anwar Haque (2008). "Random-Forests-Based Network Intrusion Detection Systems." In: *IEEE Transactions on Systems, Man, and Cybernetics* 38.5, pp. 649–659 (cit. on p. 89).

DECLARATION

I hereby confirm that the submitted thesis with the title "Improving the Capabilities of Distributed Collaborative Intrusion Detection Systems using Machine Learning" has been done independently and without use of others than the indicated aids. I assure that I have not previously or concurrently applied for the opening of a promotion procedure with the doctoral thesis submitted here.

Darmstadt, August 21, 2019

Carlos García Cordero,
August 21, 2019