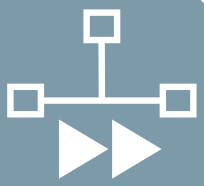
Security in High-Bandwidth Networks

Dissertation zur Erlangung des Doktorgrades Dr. rer. nat.
der Fakultät für Ingenieurwissenschaften, Informatik und Psychologie
der Universität Ulm



Vorgelegt von
Thomas Lukaseder
aus Lauingen (Donau)

Institut für Verteilte Systeme
Universität Ulm, Deutschland

SECURITY IN HIGH-BANDWIDTH NETWORKS

THOMAS BASTIAN LUKASEDER

SECURITY IN HIGH-BANDWIDTH NETWORKS

DISSERATION

zur Erlangung des Doktorgrades Dr. rer. nat. der
Fakultät für Ingenieurwissenschaften, Informatik und Psychologie der
Universität Ulm

vorgelegt von

THOMAS BASTIAN LUKASEDER

aus Lauingen (Donau)

2020

Amtierender Dekan:
Prof. Dr.-Ing. Maurits Ortmanns

Gutachter:

Prof. Dr. rer. nat. Frank Kargl, Universität Ulm
Prof. dr. ir. Aiko Pras, Universiteit Twente
Prof. Dr.-Ing. Dr. h.c. Stefan Wesner, Universität Ulm

Tag der Promotion
05.06.2020

ABSTRACT

Ever-increasing bandwidth in networks presents a challenge to security mechanisms as the amount of traffic (following Gilder's law) increases faster than the computational power (following Moore's law). This continuous increase in the amount of data not only impedes the effort to analyze the data in firewalls or Intrusion Detection Systems, but it can also be exploited by attackers to achieve ever stronger attacks. Moreover, testing network security mechanisms in high-bandwidth networks presents a challenge in itself as common testing tools are neither designed to produce nor to analyze such a vast amount of traffic.

In this thesis, firstly, we look into testing of network applications, devices, and algorithms in high-bandwidth networks as a challenge in and of itself. We analyze traffic, build a network testing framework, and provide test data sets as groundwork for the other parts of this thesis.

Following these insights, we work on improving security mechanisms to tackle the challenges of high-bandwidth networks. Hereby, we focus on two commonly used security mechanisms found in today's networks: Intrusion Detection Systems (IDS) and Mitigation Systems for Distributed Denial-of-Service (DDoS) attacks and investigate the impact of rising network traffic on their performance.

We look into ways to raise IDS throughput through hardware-supported parallelization of regular expression matching. Matching regular expressions is a key component of the payload analysis in IDS and presents a major bottleneck for their throughput.

Moreover, we present a framework able to detect DDoS attacks, identify attacking clients, and defend successfully against attacks. The system entails improvements in these areas with a particular focus on identifying slow DDoS attackers and defense against reflective attacks.

The software developed, the data sets produced, and the insights gained in this work can help researchers, network administrators, and developers improve network security mechanisms and defend their networks more reliably against attacks.

ZUSAMMENFASSUNG

Die ständig wachsende Bandbreite in Netzen stellt eine Herausforderung für die Sicherheitsmechanismen dar, da die Menge des Datenverkehrs nach Gilder's Gesetz schneller steigt als die Rechenleistung nach Moore's Gesetz. Diese kontinuierliche Zunahme der Datenmenge erhöht nicht nur den Aufwand zur Analyse der Daten in Firewalls oder Intrusion Detection Systemen, sondern kann auch von Angreifern genutzt werden, um immer stärkere Angriffe zu erzielen. Darüber hinaus stellt das Testen von Sicherheitsmechanismen in Netzen mit hoher Bandbreite eine Herausforderung an sich dar, da Testwerkzeuge weder dazu ausgelegt sind, eine so große Menge an Datenverkehr zu produzieren noch zu analysieren.

In dieser Arbeit untersuchen wir zunächst das Testen von Anwendungen, Geräten und Algorithmen in Netze mit hoher Bandbreite als eine Herausforderung an sich. Wir analysieren den Datenverkehr, bauen ein Netzwerk Testframework und stellen Testdatensätze zur Verfügung die uns im weiteren Verlauf der Arbeit nützen.

Folgend den erlangten Erkenntnissen arbeiten wir an der Verbesserung der Sicherheitsmechanismen, um die Herausforderungen von Netzen mit hoher Bandbreite zu bewältigen. Dabei konzentrieren wir uns auf zwei häufig verwendete Sicherheitsmechanismen, die in heutigen Netzwerken zu finden sind: Intrusion Detection Systeme (IDS) und Abwehrsysteme von Distributed Denial of Service (DDoS) Angriffen und untersuchen die Auswirkungen des steigenden Netzwerkverkehrs auf ihre Leistung.

Wir untersuchen Möglichkeiten, den IDS-Durchsatz durch die Parallelisierung von Regular Expression Matching durch Hardwareunterstüzung zu erhöhen. Das Matching von regulären Ausdrücken ist eine Schlüsselkomponente der Payload-Analyse im IDS und stellt einen großen Engpass für deren Durchsatz dar.

Darüber hinaus stellen wir ein Framework vor, das in der Lage ist, DDoS-Angriffe zu erkennen, Angreifer zu identifizieren und sich erfolgreich gegen Angriffe zu verteidigen. Das System bringt Verbesserungen in diesen Bereichen mit sich, mit besonderem Fokus auf die Identifizierung von slow DDoS-Angreifern und die Abwehr von Reflective Angriffen.

Die entwickelte Software, die produzierten Datensätze und die dabei gewonnenen Erkenntnisse können Forschern, Aministratoren und Entwicklern helfen, ihre Sicherheitsmechanismen in Netzen zu verbessern und ihre Netze zuverlässiger gegen Angriffe zu schützen.

PUBLICATIONS

Some earlier versions of the material presented in this thesis have previously appeared in the following publications:

[1] **T. Lukaseder**, L. Bradatsch, B. Erb, R. W. van der Heijden, and F. Kargl. "A Comparison of TCP Congestion Control Algorithms in 10G Networks." In: *IEEE 41st Conference on Local Computer Networks (LCN)*. Oct. 2016. DOI: 10.1109/LCN.2016.121 (cit. on pp. 7, 37, 43, 55, 57, 59, 60, 62, 63).

[2] **T. Lukaseder**, L. Bradatsch, B. Erb, and F. Kargl. "Setting Up a TCP Benchmarking Environment—Lessons Learned." In: *IEEE 41st Conference on Local Computer Networks (LCN)*. Oct. 2016. DOI: 10.1109/LCN.2016.32 (cit. on pp. 7, 37, 39, 40, 43).

[3] **T. Lukaseder**, J. Fiedler, and F. Kargl. "Performance Evaluation in High-Speed Networks by the Example of Intrusion Detection Systems." In: *11. DFN-Forum Kommunikationstechnologien*. 2018 (cit. on p. 8).

[4] **T. Lukaseder**, A. Hunt, C. Stehle, D. Wagner, R. van der Heijden, and F. Kargl. "An Extensible Host-Agnostic Framework for SDN-Assisted DDoS-Mitigation." In: *IEEE 42nd Conference on Local Computer Networks (LCN)*. Oct. 2017. DOI: 10.1109/LCN.2017.103 (cit. on pp. 8, 158, 178, 181).

[5] **T. Lukaseder**, L. Maile, B. Erb, and F. Kargl. "SDN-Assisted Network-Based Mitigation of Slow DDoS Attacks." In: *EAI International Conference on Security and Privacy in Communication Networks (SecureComm)*. Aug. 2018. DOI: 10.1007/978-3-030-01704-0_6 (cit. on pp. 9, 79, 166, 198, 199).

[6] **T. Lukaseder**, L. Maile, and F. Kargl. "SDN-Assisted Network-Based Mitigation of Slow HTTP Attacks." In: *1. KuVS Fachgespräch Network Softwarization – From Research to Application*. 2017. DOI: 10.15496/publikation-19543 (cit. on pp. 9, 166).

[7] **T. Lukaseder**, K. Stölzle, S. Kleber, B. Erb, and F. Kargl. "An SDN-based Approach For Defending Against Reflective DDoS Attacks." In: *IEEE 43rd Conference on Local Computer Networks (LCN)*. Oct. 2018. DOI: 10.1109/LCN.2018.8638036 (cit. on pp. 9, 170).

[8] **T. Lukaseder**, S. Ghosh, and F. Kargl. "Mitigation of Flooding and Slow DDoS Attacks in a Software-Defined Network." In: *IEEE 43rd Conference on Local Computer Networks (LCN), Demo Track*. Oct. 2018 (cit. on p. 8).

[9] L. Bradatsch, **T. Lukaseder**, and F. Kargl. "A Testing Framework for High-Speed Network and Security Devices." In: *IEEE 42nd Conference on Local Computer Networks (LCN)*. Oct. 2017. DOI: 10.1109/LCN.2017.91 (cit. on pp. 7, 65).

[10] F. Engelmann, **T. Lukaseder**, B. Erb, R. van der Heijden, and F. Kargl. "Dynamic packet-filtering in high-speed networks using NetFPGAs." In: *IEEE Third International Conference on Future Generation Communication Technologies (FGCT 2014)*. Aug. 2014. DOI: 10.1109/FGCT.2014.6933224 (cit. on pp. 8, 110).

CONTENTS

LIST OF FIGURES

I

INTRODUCTION

1 INTRODUCTION

1.1 MOTIVATION

Institutional networks such as university networks, business networks, or networks of other public institutions are commonly connected to the Internet through one or several routers through a firewall-secured connection (i. e., perimeter security). Firewalls *proactively* filter packets and block clients based on predefined rules. These rules can be based on policy decisions or based on security considerations. Within the networks, switches are routing packets to the network edges where clients are connected either via Ethernet connections or through a WiFi access point.

Even in well-secured networks with good perimeter security, it cannot be assumed that there are no attackers or compromised devices within the network [129, 175]. In the case of university networks such as the Ulm University network infrastructure, end-user devices are often the user's property and cannot be secured by the network operators. An increasing amount of devices in the network are mobile and are commonly used in other networks, which are often unsecured or at least poorly secured. Moreover, one cannot rely on laymen end-users to sufficiently protect their devices. In addition, there are guest networks and attacks from the Internet that might penetrate the firewall. In order to be able to achieve acceptable network security despite insufficient perimeter security, Intrusion Detection Systems (IDS) are often deployed in the networks. They can identify threats, for example, in the form of malware, worms, or brute-force access attempts. IDS analyze the traffic and *reactively* report threats to the network administrators. In the case of an Intrusion Prevention System (IPS), the system itself also tries to defend against the attacks automatically when attacks where identified. One special case of IPS are systems mitigating Distributed Denial-of-Service (DDoS) attacks. DDoS attacks are among the most widespread and common network-based attacks. While other network-based attacks can often be observed analyzing single packets or single flows, observing DDoS attacks requires information about the mixture of traffic within the network. This is often achieved by analyzing flow information for the whole network infrastructure instead of analyzing single client connections.

These mechanisms — firewalls, IDS, and IPS — should assure a secure network environment. However, they face many challenges. In many networks but especially research networks, where minimal re-

strictions already could mean severe limitation for researchers, security often has to take a back seat to full availability of the services. In the meantime, the threat landscape in these networks changes. Networks become more complex, dynamic, and faster. Ever-increasing bandwidth in networks presents a challenge to security mechanisms as the amount of traffic increases faster than the computational power. While Moore's law states that computational power increases 5-fold over 4 years [224], Gilder's law states that bandwidth increases in the same period 32-fold [131]. This continuous increase in the amount of data and growing gap between processing capabilities and throughput not only impedes the effort to analyze the data in firewalls or Intrusion Detection Systems, but it can also be exploited by attackers to achieve ever stronger attacks as the continues rise of DDoS throughput shows [297].

1.2 RESEARCH QUESTIONS

Facing these challenges in network security, we want to focus on one overarching research question in this work:

> How can we improve security mechanisms while being faced with increasing amounts of data and attack complexity?

Due to their current and increasing complexity, reactive security mechanisms such as IDS are predominantly affected by the changing threat landscape. Therefore, we focus on these mechanisms. Denial-of-Service attacks are a class of network attacks that majorly rely on throughput. Following the current trend, their importance will probably continue to grow, even beyond today's high level. We will, therefore, focus in particular on the defense against these attacks. Early on in our research, we realized that the necessary tests for such mechanisms at high bandwidths are insufficient for our work. Before all else, we had to assure that our security mechanisms could be evaluated, resulting in our own testing model and infrastructure.

All in all, the impact of rising bandwidth in networks affects security applications in the area of testing these applications, analyzing traffic in real-time, and defending against increasingly powerful attacks. It is, therefore, the goal of this thesis to present improvements to network security in high-bandwidth networks with the following three central research questions:

1. How can realistic traffic be modeled and produced to test network mechanisms?

Intrusion Detection Systems and Firewalls are enormously important for securing networks. Test infrastructure is needed to successfully verify whether these systems provide the security we expect from them. Testing network security mechanisms in *high-bandwidth*

networks presents a challenge in itself as common testing tools are neither designed to produce nor to analyze such a vast amount of traffic. These tests need to evaluate performance in terms of accuracy and throughput to ensure that the systems reliably identify attacks while keeping false positives at a minimum. Testing security applications requires both attack traffic and benign traffic to see how network security mechanisms are handling different mixtures of traffic. The security mechanisms need to be able to discern benign from attack traffic and must be able to reach their required performance regardless of background traffic composition. The traffic must be as close as possible to reality to assure that these results can be transferred to production networks. This part predominantly focuses on an analysis of testing mechanisms and data sets that are typically used for these evaluations. The evaluation aspect in *high-bandwidth networks* is also supported by an in-depth use case evaluation of TCP congestion control algorithms. These works then lead to a framework for testing network devices and algorithms that, in turn, is used to answer the other research questions.

2. Can hardware-based acceleration help Intrusion Detection Systems to obtain the throughput needed in future networks?

IDS throughput relies heavily on their performance concerning the live analysis of traffic. Core aspects of this analysis are string matching and regular expression matching. Increasing the performance of these mechanisms would improve the throughput of IDS and, therefore, their applicability in future networks. One method of acceleration is the use of parallel processing in specialized hardware modules. Some research was done in this area. However, a definitive answer if hardware-based acceleration of IDS is feasible is missing. To answer this question, we highlight various methods to accelerate IDS with hardware modules based on FPGAs and GPUs. This is followed by an exemplary implementation of the GPU-based mechanism.

3. How can the mitigation of Distributed Denial-of-Service Attacks be improved?

We give special attention to one use case of IDS — the mitigation of DDoS attacks. Most DDoS attacks require high throughput and are, therefore, especially interesting for attackers to utilize in high-bandwidth networks. Unlike other network-based attacks where full traffic analysis is necessary, oftentimes for DDoS attacks, only statistical analysis is needed making it easier to detect the attacks. However, the network itself suffers under the load produced by many DDoS attacks challenging the mitigation system. As part of this work, a new framework is built. This framework enables us to evaluate mechanisms to (1) detect attack, (2) identify attackers, and (3) defend the network against the attacks.

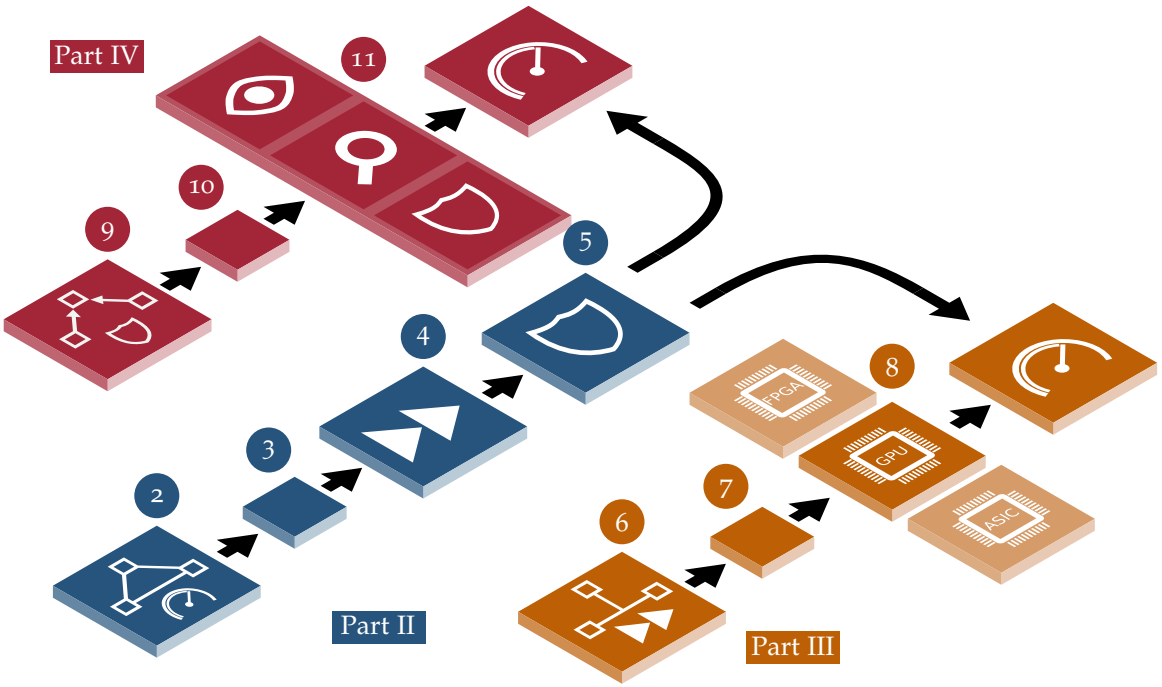


Figure 1: Roadmap of this thesis. The Arabic numerals represent the chapters while the Roman numerals represent the parts. All parts begin with an introductory survey (Chapters 2, 6, 9) followed by a research statement (Chapters 3, 7, 10) leading into the main contributions (Chapters 4, 5, 8, 11).

1.3 OVERVIEW AND CONTRIBUTIONS

In the following, the core results and research topics of this work are summarized. Figure 1 contains the basic structure of this thesis. In Part II the focus lies on the conceptual design and implementation of a network testing framework as a basis for testing the resulting implementations of the other two parts. In a second step, two commonly used security mechanisms found in today's networks are discussed in Part III and Part IV: Intrusion Detection Systems (IDS) and Mitigation Systems of Distributed Denial-of-Service (DDoS) attacks. Research in these areas focuses on the impact of rising network traffic on their performance and improved mechanisms both for better high-bandwidth network performance and higher accuracy.

Part II: Network Testing (Chapters 2 to 5)

The core of this contribution is the *General Network Testing Framework (GPNTF)*, which aims to improve the methodology of network testing, i. e., testing of network devices, protocols, and software for network operations. To reach the point of implementing this framework, two steps needed to be undertaken. For one, general network tests at high throughput needed to be achieved. In a second step, we make

the jump from general tests of network devices and protocols to specialized security tests.

SURVEY ON NETWORK TESTING An extensive analysis was conducted on network testing, in particular on data sets used for testing of network devices and mechanisms. The analysis was focused predominantly on Intrusion Detection System evaluation and the evaluation of DDoS mitigation systems. This contribution is documented in Chapter 2.

TCP CONGESTION CONTROL ALGORITHMS IN HIGH-BANDWIDTH NETWORKS As a first step in the area of network testing, we chose a non-security related, however common and relevant issue in networks: The evaluation of TCP congestion control algorithms. Many evaluations have been conducted in lower bandwidth networks up to 1 Gbit/s, and when introducing a new protocol, the protocol designers often evaluated their algorithms against the then state of the art. With these evaluations as a blueprint, this use case served as a starting point of our endeavors in the area of high-bandwidth network testing. The area is well researched, which gives us a good starting point while independent evaluation of common mechanisms at high throughputs was missing. Common pitfalls and challenges where identified and configurations necessary to achieve 10 Gbit/s with commodity hardware were documented. 10 Gbit/s was chosen as for single flow applications, 10 Gbit/s is common in data centers and starts to become common even in some desktop systems while higher throughput is still in the experimental stage [146]. This contribution is documented in Chapter 4.

Results from this chapter have been published at IEEE LCN 2016 [1, 2].

ANALYSIS AND PRODUCTION OF NETWORK TRAFFIC Reaching sufficient throughput is a necessary albeit insufficient step to build a complete testing environment for security applications. In addition, a realistic composition of traffic is necessary to achieve convincing analyses. The number of connections per client, the quantity of clients in the network, connection lengths, or the variance of traffic patterns are necessary to take into account for many network testing applications. Benign traffic is important for testing the usual networking equipment without security context and testing security applications to measure false-positive rates. In addition to benign traffic, for security applications test, malicious traffic is needed to induce into the network. For testing Intrusion Detection Systems, different kinds of intrusions need to be possible to produce. From SSH brute force attacks, botnet traffic, worms, to full Distributed Denial-of-Service attacks — a wide range of features needs to be available. As a result of this work, we present the General Purpose Network Testing Framework (GPNTF). Chapter 5 contains a detailed description of this work.

Results from this chapter have been published at IEEE LCN 2017 [9].

Part III: Acceleration of Intrusion Detection Systems (Chapters 6 to 8)

Matching regular expressions is a key component of the payload analysis in IDS and presents a major bottleneck for their throughput. In this part, raising the throughput of IDS through hardware-supported parallelization of regular expression matching is the focus.

SURVEY ON INTRUSION DETECTION SYSTEMS A survey on Intrusion Detection Systems and how these are commonly accelerated was conducted. Chapter 6 documents the results of this analysis.

Results from this chapter have been published at IEEE FGCT 2014 [10] and at the 11th DFN-Forum 2018 [3].

ANALYSIS OF HARDWARE ACCELERATION APPROACHES The first step was to identify and evaluate concepts for the acceleration of IDS. We identified three different concepts based on FPGAs, ASICs, and GPUs. Our analysis showed that GPUs have the most potential for the acceleration of IDS, leading to the decision to focus on this acceleration method going forward. Following this decision, the GPU-acceleration concept was refined, implemented, and evaluated. Chapter 8 describes our results in this area.

Part IV: Mitigation of DDoS Attacks (Chapters 9 to 11)

In the area of DDoS attack mitigation, we considered many different attacks and many defense mechanisms against them. At that, we focused on the use case of mitigation of the attacks solely within the network infrastructure. That means we do not consider any help from the attack target. This represents a typical use case in our research network as servers are often administered by research institutes or schools directly. Hardware and software can be quite diverse. Administrators might not be available during attacks or might not be trained to help. Simultaneously, administrators from the research network do not have access to the target systems. Therefore, independent of attack, we need to be able to mitigate attacks within the network infrastructure.

SURVEY ON DDOS ATTACKS AND MITIGATION Chapter 9 contains an extensive survey on both Denial-of-Service attacks commonly found in today's Internet and current approaches on how to tackle them.

The framework has been published at IEEE LCN 2017 [4] and IEEE LCN 2018 [8].

PRESENTING A DDOS MITIGATION FRAMEWORK We build a framework based on Software-Defined Networking (SDN) technology to facilitate our mitigation presented in Chapter 11. SDN was chosen as the technology offering the flexibility necessary to build a functioning prototype quickly. However, none of the mechanisms necessarily need SDN to function in principle. Mitigation of DDoS

attacks typically consists of three main steps: detection of attacks, identification of attackers, and defense against the attack. In all three areas, the framework contains research contributions.

detection

There are many ways to detect attacks. Especially in recent years, a multitude of detection mechanisms have been proposed, for instance, based on entropy calculations of flow metrics. We conducted an extensive analysis of how these mechanisms perform in research networks. Common methods to detect DDoS attacks in view of high sampling rates, dependent on network sizes and types, are undertaken based on data from the research network Baden-Württemberg (BelWü) and other public data sets.

identification

Results were published at EAI SecureComm 2018 [5] and KuVS Fachgespräch 2017 [6].

In the identification realm, the framework is able to identify attackers running vastly different attacks including HTTP and TLS flooding but also slow HTTP attacks. The flooding attack clients are identified by allocating a score to the clients in the network based on the estimated load they cause on the target. Improvements were made in the area of identification of attackers running different slow HTTP attacks presenting a new identification scheme based on the combination of the packet rate and the packet distance uniformity. With slowloris-ng, a new slow attack tool was implemented and published to challenge slow HTTP attack identification schemes.

defense

Results were published at IEEE LCN 2018 [7].

Furthermore, the framework contains effective means to defend against DDoS attacks. For one, we present a system to block individually identified clients when faced with application-layer attacks. Furthermore, a special mechanism presents improvements to how traffic can be filtered in networks when faced with UDP-based reflective attacks. The mechanism is easy and fast to deploy, scalable, and transparent to the attack victim.

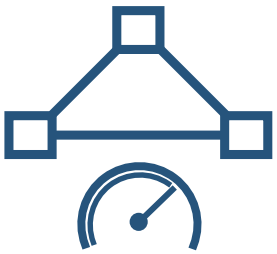
1.4 OPPORTUNITIES — THE RESEARCH PROJECTS, AND THEIR RESOURCES

This work originated in the context of the *bwNET100G+* research project funded by the *Ministry of Science, Research and the Arts Baden-Württemberg (MWK)*. This project is unusual in its design as it combines network operations of three universities in Baden-Württemberg (Tübingen University, Ulm University and the Karlsruhe Institute of Technologies), the state-owned Internet service provider of the research network in the state *Baden-Württemberg extended LAN* (BelWü), and three research institutes at the three aforementioned universities. The stated goal of the research project is *"research and innovative services for a flexible 100G-network in Baden-Württemberg"*[1]. Input from the operations side shall ensure that the research done in this project is practically relevant in modern networks, while access to the infrastructure and experience of the operations offers exceptional opportu-

1 https://bwnet100g.de

nities to the researchers to investigate practice-oriented research questions. Three topics build the main focus of the project. For one, methods for flexible and intelligent network services mainly researched at *Tübingen University* with the focus on the application of Software-Defined Networking in research networks. Secondly, data transport in high-bandwidth networks as a prime responsibility of the *Karlsruhe Institute of Technology*. *Ulm University* focused on the security aspects of high-bandwidth networks. The results of this endeavor at Ulm University can be found in these pages.

Another research project based on the same fundamental principle is the *bwNetFlow* research project funded by the same ministry. *Ulm University* and *BelWü* are also part of the main contributors to this project. Its primary focus lies in implementing a flow monitoring infrastructure for billing purposes for peering partners of the research network. However, such an infrastructure opens up additional opportunities for researchers in the networks and network security area: Attack detection mechanisms can be tested in a live network infrastructure. Close relations with this research project and access to their data opened up possibilities of extensive data analysis in the research network of Baden-Württemberg documented in this thesis.



II

NETWORK TESTING

2 INTRODUCTION TO NETWORK TESTING

Testing network devices and network protocols (i. e., performance and accuracy tests) require the simulation or emulation of the real network infrastructure as close as possible. New network devices and protocols need to be tested to ensure that they fulfill their design goals. Testing in a production network is usually not feasible as the functionality of the network could be impeded by faulty, untested equipment or errors in the protocol design. Furthermore, repeatability of tests and results is usually necessary to test improvements or compare different devices. Therefore, realistic, controlled testing environments are a necessity. Building a testing environment that resembles the real network's properties is an important non-trivial task, especially with the focus on high-bandwidth networks.

However, assurance of high throughput is not the only requirement for comprehensible testing infrastructure. A testing framework for networks has to emulate several parts of the network infrastructure. For one, the traffic has to show typical features of traffic in the specific network type under consideration. This contains traffic patterns (e. g., size and length of flows, the prevalence of certain protocols), user behavior, utilization of the network resources (e. g., is the network overloaded? Is there congestion in the network?). Therefore, designing tests always encompass at least two necessary steps:

1. Choosing programs, test scenarios, topologies that can be used to perform the tests.
2. Choosing data that contains all necessary features of the network of operation, where the list of specific features is dependent on the test scenario and test objective.

Programs to facilitate tests are abundant and manifold (as we will show in this chapter). However, so are the operational areas and requirements of network devices and protocols. Some tests require programs focused on high traffic throughput; others focus more on precision. Some require specific network features common only in specialized environments (e. g., in automotive networks or cyber-physical systems), others need traffic patterns that are as similar to commonly found patterns as possible.

One common way to assure that traffic closely resembles the network traffic of the operational network is to use recordings of similar production networks. That way, features such as usual user behavior are featured in the data sets without the need to analyze it formally. These data sets are a cornerstone for testing network devices and

protocols. They assure repeatability and comparability between tests as the same data set can be used for many tests. However, there are some limitations. One of them is obtaining the data sets in the first place. This can be a hassle as data protection laws limit options significantly, especially concerning the publication of the data sets. Unpublished data sets are of limited use as they cannot ensure the reproducibility of evaluations across working groups and the full extent of the research community. However, many research institutions, over time, published and continue to publish valuable data sets that enable testers to perform comparable tests, which we will analyze in the following.

There are several cases where data sets cannot be used to build meaningful tests. For example in case no recorded data set is obtainable, or tests require features that are under development or available but not live in any production network yet (such as new protocols), or special variations of the network need to be tested. Some network tests require the device under test (DUT) to modify the network behavior and therefore changing the features of the data live during testing, for instance, when an Intrusion Prevention System (IPS) actively blocks traffic in the network. If a reciprocation between DUT and the traffic exists, static data sets reach the limits of their usefulness. In these cases, traffic has to be emulated as close to reality as possible. A realistic emulation first requires precise models of the real traffic that then have to be closely emulated.

In this chapter, the current state of network testing will be laid out. In context of this thesis, network tests are discussed in two distinct security related use cases for parts III and IV:

1. Performance tests of IDS and DDoS mitigation systems.
2. Accuracy tests of DDoS mitigation systems.

We identified two necessary steps to achieve network tests that are able to fulfill our requirements:

1. We need to be able to achieve high throughput.
2. We need realistic traffic, be it recordings or simulated traffic.

For this, data sets, programs, and common testing methodologies will be analyzed in the following.

2.1 DATA SETS

Christian Forst [20] has contributed to this section with his master thesis.

Finding the right data set can be tricky as there is a large variety but no central resource that could help find them. However, some online resources provide information on data sets that can be used for different purposes. In the following, we distinguish between primary

sources hosting data sets themselves and secondary sources providing information on data sets and lists of links to these data sets and their primary source.

Primary sources are often hosted by the ones creating or recording the traces. The University of New Brunswick, Canada, offers a multitude of different data sets[1], some of them for testing network devices such as IDS. In the United States, many data sets are available through the IMPACT Cyber Trust portal[2] operated by the Department of Homeland Security (DHS). Access is granted only from DHS-approved locations (United States, Australia, Canada, Israel, Japan, Netherlands, Singapore, and the United Kingdom). However, access to some data sets can be obtained from other locations on request. Contagiodump[3] is a blog posting malware samples regularly. The DEFCON CTF Archive[4] provides packet captures from their CTF events with numerous attacks. Similarly, the Mid-Atlantic CCDC[5] and the anti Malware engineering Workshop (MWS)[6] — other annual competitions — also provide network traces of their events. The Internet Traffic Archive[7] hosts a variety of older network traces, mostly from the 1990s. Malware Traffic Analysis is a blog and *"source for pcap files and malware samples"*[8] and offers a long list of contemporary attack traffic. The site is very active and provides frequent updates of new traffic traces. The Simple Web[9] — curated at the University of Twente — provides tutorials, podcasts, information, and network traces relevant to Internet management. The traces are available online under free licenses.

Apart from the sites curated by the ones creating or recording the traffic, there are also sharing sites that allow uploads of data sets. DDoSDB[10] offers a large variety of meta-information on real, captured DDoS and DoS attacks that can be used to simulate realistic attacks. Kaggle[11] is a data set sharing site not specialized in network traces but all kinds of data (e. g., Schengen Visa Statistics of 2017/2018). Network traffic data is a small part of the vast collection, but there are some data sets worthy of attention. OpenML[12] is a sharing site focused on machine learning related data sets, including network data analysis.

1 https://www.unb.ca/cic/datasets/index.html
2 https://www.impactcybertrust.org
3 http://contagiodump.blogspot.com/
4 https://www.defcon.org/html/links/dc-ctf.html
5 https://maccdc.org/
6 https://www.iwsec.org/mws/2018/en.html
7 ftp://ita.ee.lbl.gov/html/traces.html
8 https://malware-traffic-analysis.net
9 https://www.simpleweb.org/wiki/index.php/Main_Page
10 https://ddosdb.org
11 https://www.kaggle.com
12 https://www.openml.org

Using a secondary source can help to put the data sets into context and also can provide a more neutral look at the data sets. Furthermore, regularly looking at these lists helps to keep up to date with new data sets being published. One of the secondary resources is SecRepo. SecRepo is a public database of *"Samples of Security Related Data"*[13] curated by Mike Sconzo. It offers a list of publicly accessible data sets for network tests focused on the security domain. Both simulated and recorded traffic can be found here, mostly in the pcap file format. AZ-Secure[14] provides a collection of links to numerous data sets. Security researcher Jason Trost operates covert.io[15] providing a list of data set links. NETRECSEC[16] curates a list of publicly available data sets relevant for network security. RIPE[17] hosts some data sets, for example, containing DNS lookups from 40 vantage points[18] and also mirrors the WITS data sets.

In addition to the resources mentioned here, many single-purpose sites were put up for other data sets discussed in the following. There is a multitude of network testing data sets with different goals, extents, and depths. They differ in their availability and accessibility. They were recorded in simulated networks or real networks — or they encompass a mixture of both. In the case of data sets with attack traffic mixed in, the types and extent of the attacks can be very different. For security device testing, it is furthermore crucial that the attack traffic in mixed data sets is identifiable as such (i. e., attacks are labeled). It is important to choose data sets fitting the target network of a device or algorithm. Otherwise, the significance of test results could be called into question.

In the following, many of the most common data sets are analyzed. With the focus of this thesis on Intrusion Detection Systems and Denial-of-Service mitigation, the list of data sets is also limited to data sets suitable for these use cases. Additional to the resources discussed in the following, some resources are providing small packet samples and captures that can help for smaller-scale analyses[19].

2.1.1 *Artificial Data Sets*

Many data sets were created artificially, with deliberately chosen attacks meant to enable complete, broad evaluations of IDS.

13 https://secrepo.com
14 https://www.azsecure-data.org
15 http://covert.io
16 https://www.netresec.com/index.ashx?page=PcapFiles
17 https://labs.ripe.net/datarepository
18 https://v6day.ripe.net/
19 e. g., https://www.openpacket.org or https://wiki.wireshark.org/SampleCaptures

- The Coburg Intrusion Detection Data Sets (CIDDS)[20] are one example of this kind of data sets. CIDDS-001 only contains simulated, benign data [261] but comes with extensive documentation [259] and was also statistically analyzed by a third party [309] for distance-based machine learning approaches. The CIDDS-002 data set was created based on the system build and documented for CIDDS-001 [260]. The data set contains both simulated benign traffic in business networks and port scan attacks. Both data sets were documented in technical reports [258, 259].

- The data sets published by the Defense Advanced Research Projects Agency (DARPA)[21] are among the best known and most widely used data sets that can be found. For the use case of Intrusion Detection System evaluation, they published a specialized data set as early as 1998 [141, 198]. The first data set of 1998 is split into a training data set and a test data set with a duration of seven and two weeks, respectively. Both sets contain typical network attacks such as network scans and DoS attacks. The DARPA data set from 1999 contains a total of 56 different attacks in 201 variations extending the ones included in the previous version. Among others, trojan horses were added as well as spam e-mails, and DoS attacks on e-mail servers. Both the training data set and the test data set are comprised of two weeks of recordings each. In 2000, DARPA published additional data sets more focused on specific use cases [141]. LLDOS 1.0 and 2.0 deal with DoS attacks on Solaris systems. The data sets not only contain the DoS attack itself, but they also include the reconnaissance and intrusion and takeover of a Solaris system before the attack starts. The Solaris system was subsequently used as an attacker. LLDOS 2.0 differs from LLDOS 1.0 by the complexity of the attacks and the stealthiness of the attacker. According to the data set publishers, the adversary in the attacks shall represent a novice attacker. The Windows NT data set published in the same year contains different attacks targeting Windows NT. Data and labeling information for all DARPA data sets are available on their homepage. While the data set is extremely extensive and, without a doubt, one of the most useful data sets to date, as the data sets were recorded at the turn of the century, it can be questioned how relevant their traffic patterns are still today. Both benign and attack traffic changed quite substantially in the last 20 years (e. g., the introduction of new protocols such as QUIC, changes in user behavior such as the rise of streaming platforms, social media, and mobile sys-

20 https://www.hs-coburg.de/index.php?id=927
21 https://www.ll.mit.edu/r-d/datasets

tems, and new attacks such as reflective DoS attacks that were unheard of 20 years ago).

- The Canadian Institute for Cybersecurity at the University of New Brunswick (UNB) published the simulated ISCXIDS2012 data set[22] [281] meant for IDS evaluation. The traffic was generated following a two-step approach. In the first step, traffic was recorded within their own network. Profiles were created on the user behavior for every user in the network, and traffic was generated based on these profiles in a second step. The profiles contain a user behavior model based on how users use HTTP, SMTP, SSH, IMAP, POP3, and FTP. With this approach, the authors aimed to produce both realistic and unanonymized traffic. In addition to this benign traffic generation, attack traffic was generated. For the Distributed Denial-of-Service attacks conducted for these data set recordings, the authors built their own IRC-based botnet software performing a simple HTTP flooding attack over a period of 60 minutes.

- The UNSW-NB data set from the University of New South Wales at the Australian Defence Force Academy (UNSW Canberra)[23] was publicly released 2015 to replace the *"unbalanced and outdated data sets"* [227] commonly used. In order to generate attacks and background traffic, a traffic generator was connected to a virtual network consisting of three virtual servers. This resulted in 175 000 records for training and 82 000 records for testing. Attacks within the records are classified as Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode, and Worms. The data set is recorded in the pcap file format. Argus and Zeek records are included with the data sets as well as a CSV documentation of the attacks together with every attack linked to corresponding CVE reports.

- The TUIDS data sets contain different, labeled attacks along with benign traffic in a simulated environment. The data sets contain network scans, an intrusion data set, and DDoS attacks [61, 133]. The attacks conducted are SYN and XMAS scans through Nmap, the DDoS attacks use Fraggle and Smurf, and the tool targa2 is used to simulate 12 different intrusion attacks. Unfortunately, the data sets are not available for download at the reported link[24] anymore.

22 https://www.unb.ca/cic/datasets/ids.html

23 https://www.unsw.adfa.edu.au/australian-centre-for-cyber-security/cybersecurity/ADFA-NB15-Datasets/

24 http://www.tezu.ernet.in/~dkb/resources.html as reported in [262]; falsely reported as http://www.tezu.ernet.in/␣dkb in the original publication.

2.1.2 *Pure Attack Traffic*

There are also data sets that only contain attack traffic. Data sets with real attack traffic recordings in the wild are scarce, and most research groups fall back to three different methods to obtain attack traffic. They simulate the attacks themselves, record traces at hacking competitions, or use honeypots in hopes that attacks on them are conducted.

SIMULATED ATTACKS have the advantage of being entirely under the control of the conductor; type and extent of the attack can be chosen. However, this also means that biases of the researchers creating the data sets remain unchecked.

- The DDoS 2016 data set [167] contains attacks conducted in the obsolete NS-2 simulation environment. It contains a UDP flood, a smurf attack, an HTTP flood, and a SQL Injection DDoS attack. The documentation of the data set is sparse, and the data set is not publicly available.
- Botnet [63] is a data set combining the ISOT, ISCX 2012, and CTU-13 data sets. The resulting data set contains botnet traffic of 16 different botnets. The traffic data is separated into a learning and testing data set.
- We did gain access to one of the data sets in the IMPACT Cyber Trust database in the following called USC/LANDER[25]. USC/-LANDER contains a reflective DDoS attack using DNS servers as reflectors. The full setup was under the control of the University of Southern California Information Sciences Institute. The attack is roughly half an hour long and targeted one machine with six reflectors from one attacker. The data set contains the traffic between reflectors and attack target; the requests sent by the attacker are not included in the traces.
- In 2013 and 2014, the ADFA sets[26] were created at the University of New South Wales with a separate focus on host-based IDS (HIDS) for Linux and Windows. The Linux system had Apache servers with PHP installed and some applications running (e. g., TikiWiki) [103–105]. An auditor applied attacks within the environment, such as password brute-force attacks on FTP and SSH, as well as privilege escalation attacks. All in all, it includes 833 cases for IDS training and 4373 for testing.

25 Scrambled Internet Trace Measurement dataset, IMPACT ID: USC-LANDER/DoS_DNS_amplification-20130617/rev5529. Traces taken 2013-06-17 to 2013-06-17. Provided by the USC/LANDER project (https://www.isi.edu/ant/lander).

26 https://www.unsw.adfa.edu.au/australian-centre-for-cyber-security/cybersecurity/ADFA-IDS-Datasets/

Each case of the Linux set is comprised of a list of system calls, which were run on the target during the attacks. In addition, ADFA created a windows-set attacking Windows XP SP 2 [103]. This version contains 365 training cases and 7601 test cases. Both sets are publicly available and contain attacks only.

- The TRAbID database contains traces of a variety of attacks conducted in a simulated environment [312]. The DDoS attacks are SYN flooding, UDP flooding, ICMP flooding executed with Hping3, a slow HTTP attack executed with Slowloris, an SMTP flood executed with Postal, and an HTTP flood executed with the Low Orbit Ion Cannon (LOIC). Network scans were conducted with Nmap. For each attack, a training and a validation set are available[27].

- The Network and Data Security Group (NDSec) of the Hochschule Fulda published NDSec-1 in 2017 [59]. The data set contains little benign traffic and focuses more on attack traffic. Among others, it contains Citadel botnet traffic, brute-force attacks on HTTP, FTP, and SSH, and flooding attacks using HTTP, SYN, and UDP. The data set contains PCAP and log files.

HACKING COMPETITIONS can be a great source to obtain attack traffic. Admittedly, the sheer extent of the attack traffic is not realistic for production network environments. Still, the extent and variety of attacks are much greater than when using one of the other methods.

- The KDD Cup is an annual competition in which algorithms have to compete on a specific topic organized by the ACM Special Interest Group on Knowledge Discovery and Data Mining (KDD). In 1999, the goal of the Cup was to develop IDS algorithms that were tested against the KDD99[28] data set. The set is available in the csv file format on the host's website and contains 4.9 million training traces and 300 000 test traces [295]. Attacks included in the data set are — among others[29] — LAND DoS, teardrop DoS, and a Smurf attack. The type of the record (e. g., "normal" or "smurf") is part of each record within the file. Despite its age, KDD99 is still used in recent publications [90, 142, 188]. When using the data set, the uneven distribution of attacks has to be considered, as a misrecognized attack with high occurrence can influence the test results negativly [204]. Furthermore, past evaluations of IDS with KDD have shown that comparisons of IDS might be hard to be performed due to the resulting low variety of recognition rates of tested IDS [204,

27 https://secplab.ppgia.pucpr.br/?q=trabid
28 https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html
29 Full list of attacks: https://kdd.ics.uci.edu/databases/kddcup99/training_attack_types

343]. An improved version of the set, the NSL-KDD, is a subset of KDD99 in which redundant records were removed to avoid the mentioned distribution issue [295]. The subset consists of 126 000 training records, and 23 000 test records. In addition, NSL-KDD was created as ARFF-file (Attribute-Relation File Format; the file format can be used for machine learning applications). However, the remaining characteristics (age, protocols, and variety of included attacks) are similar to KDD99.

- During similar competitions in 2000 and 2002 at the DEFCON 8 and 10, two data sets were created and published by *the Shmoo Group*[30] containing only attack traffic during the competitions. DEFCON-8 contains port scanning and buffer overflow attacks, DEFCON-10 contains privilege escalation attacks, port scans, and FTP by telnet protocol attacks [280].
- Sangster et al. analyzed whether hacking competitions can be a good source for attack traffic recordings [270] and, at the same time, publish their own CDX data set recorded at their hacking competition[31]. Snort IDS logs, domain name service logs, and web server logs complement the labeled traffic recordings of the 2009 four day event.

While the records of these data sets are extensive and many labeled attacks are recorded in the networks, the fact that the networks were set up for an event and do not contain any usual traffic patterns of normal users limits their application area.

HONEYPOTS are a great way to record random attacks on the Internet targeted at nobody specific (i. e., non-targeted attacks). Scans, botnet traffic, and brute-force attacks can be observed and recorded. However, this approach is limited to non-targeted attacks, targeted attacks against honeypots are rare, and as honeypots are often easily identifiable by a human, attacker behavior of targeted attacks on a honeypot might not be reflective of real attacker behavior.

- Kyoto University set up honeypots and recorded the Kyoto data set between November 2006 and December 2009[32]. As the data was recorded at honeypots, manual labeling and anonymization were not necessary. However, the records do not contain benign traffic patterns of regular users. The honeypots are — among others — a Solaris 8 machine, an unpatched Windows XP, and a Nepenthes honeypot. Darknet sensors, a mail server, a web crawler, and another Windows XP machine for evaluations completed the setup.

30 http://www.shmoo.com/
31 https://westpoint.edu/centers-and-research/cyber-research-center/data-sets
32 https://www.takakura.com/Kyoto_data/

- Sperotto et al. [286] from the University of Twente published a data set for flow-based IDS testing in 2009. They collected their traffic on one honeypot machine, collecting 14.2 million flows in six days, of which 98% are labeled. The honeypot ran OpenSSH, Apache web server, and FTP. In addition to the flow recordings, the logs of these services were also saved and published. The data set mainly contains brute-force login attempts, both on the target and from the target to other services. No Denial-of-Service attack traffic can be found in this data set. The data set is available on e-mail request to the authors.

- Santanna et al. [271] analyzed DDoS-as-a-Service providers (a.k.a. Booters) based on real attacks they bought from the providers attacking a honeypot they set up. The recordings of nine booter attacks (7 DNS-based and 2 CharGen-based reflective attacks) are available online at Simple Web[33].

2.1.3 *Real Traffic Recordings*

Some institutions provide recordings of traffic within their networks. Due to privacy regulations, all of these data sets are anonymized and do not provide real IP addresses. These data sets can be a great source to see how networks and their users behave and how this changes over time. As the data is mostly obtained in usual operations, there can be no guarantee that the data sets are attack free.

- The Center for Applied Data Analysis (CAIDA)[34] yearly publishes recordings of their monitoring locations at several large Internet Service Providers in the United States [87]. Between 2008 and 2014, the data sets contained traffic traces of their equinox-chicago and equinox-sanjose monitors of high-bandwidth Internet backbone links. From 2015 to 2016, the data sets contain only their equinox-chicago monitor. From 2018, the traces contain traffic from the equinox-nyc monitor. The capture was suspended at the former monitoring points when these were upgraded to 100 Gbit/s, and the CAIDA hardware could no longer capture the traffic reliably. The first data set of 2008 contained one day of anonymized (CryptoPan, prefix-preserving), bidirectional traffic. First, the data sets contained traffic from each month of the year. Due to storage constraints, newer data sets only contain quarterly recordings. Security incidents in the data set are possible but not documented. The data sets do not provide special features for Intrusion Detection Sys-

33 https://www.simpleweb.org/w/index.php/Traces#Booters_-_An_analysis_of_DDoS-as-a-Service_Attacks

34 https://caida.org

tem evaluations but can be seen as very typical Internet traffic due to their size and recording locations.

- The WIDE Project (Widely Integrated Distributed Environment)[35] is a Japanese organization founded by three Japanese universities and used to run the .jp top-level domain (taken over by the Japan Registry Services Co., Ltd.; JPRS). It currently lists over 100 mainly Japanese sponsors. The Measurement and Analysis on the WIDE Internet Working Group (MAWI) provides daily anonymized traffic data of the WIDE project[36] of one sample point in operation since July 2006. Every day, the upstream data from WIDE to the upstream ISP is recorded between 14:00 and 14:15 local time. Additionally, on special occasions (such as the *"a Day in the Life of the Internet"* project) longer recordings of up to 72 hours were recorded and published. At the time of writing, the newest, longer trace was recorded on the 9th and 10th of April 2019 (48 hours).

- The Lawrence Berkeley National Laboratory (LBNL) and International Computer Science Institute (ICSI) published data sets from October 2004 to January 2005[37]. The anonymized traces are available in the pcap file format and do not contain payload data. The data contains the traffic of several thousand internal hosts. No known attacks are contained in the files.

- Waikato Internet Traffic Storage (WITS) from the WAND Network Research Group[38] provides long traffic traces from the late 1990s and noughties. The data was collected in the network of the University of Auckland and at unnamed New Zealand ISPs. The length of the recordings is immense. As an example, the Waikato I trace is nearly 620 days long.

- The *Comprehensive, Multi-Source Cyber-Security Events* data set by Kent [170] contains 58 consecutive days of network flows metadata and event data[39] from Windows-based authentication, process start and end events on several Windows machines, and compromise events in separate files.

- The University of Brescia published the UNIBS 2009 data set as part of their research into ground truth data collection [135]. The *Cryptography-based Prefix-preserving Anonymization (Crypto-PAn)* anonymized traces were recorded through SSH tunnels at three network nodes on the network, which allowed the researchers to correlate network traces with the encrypted ses-

35 http://www.wide.ad.jp/
36 https://mawi.wide.ad.jp/mawi/
37 https://www.icir.org/enterprise-tracing/download.html
38 https://wand.net.nz/wits/catalogue.php
39 https://csr.lanl.gov/data/cyber1/

sions on the machines. The data set includes the outcome of a deep packet inspection analysis and the application responsible for a flow as returned by gt, the Ground Truth Software Suite[40]. The traces are available upon request[41].

- The Unified Host and Network Data Set by the Los Alamos National Laboratory (LANL) [304] is a publicly available NetFlow-based data set[42] of the LANL network combined with event logs originating from network nodes running in the network. The recording was done over a time frame of 90 days (89 days of flow data due to missing data on the first day).

- The CICIDS2017 data set is based on real traffic recordings but was enriched with simulated attack traffic. After they published the CICISCX data set, the Canadian Institute for Cybersecurity at the University of New Brunswick conducted a survey on data sets commonly used in IDS testing [280] in 2017. They concluded, that none of the data sets satisfy their requirements (including their own ISCX data set), and published their second IDS evaluation focused data set called CICIDS2017[43]. The data set features five days of traffic in pcap file format. The first day (Monday) only contains benign traffic recorded at their offices. The other days additionally include different kinds of attack traffic: Brute-Force FTP, brute-force SSH, web attacks (brute-force, XSS, SQL injection), several infiltration attacks, Botnet traffic, different DoS attacks (slowloris, Slowhttptest, DoS Hulk, and DoS Goldeneye), and a small DDoS attack with three attackers (apparently conducted using the Low Orbit Ion Cannon (LOIC), although documented as LOIT). Unfortunately, the recording of the traffic was conducted behind a NAT, which means that the three different attackers in the DDoS attack cannot easily be distinguished as all attack traffic seemingly comes from one machine. This and the small extent of the attack limit the usability of the data set for DDoS research but still offers a useful data set for IDS evaluations.

- The CTU-13 data set is aimed at botnet detection research [127]. Virtual machines were deliberately infected by botnets, traffic to and from these bots was recorded along with benign background traffic and labeled accordingly. Bots sending SPAM e-mails were part of the data set alongside bots conducting DDoS attacks (UDP and ICMP based) and several others. The botnet software was Neris, Rbot, Virut, Menti, Sogou, Murlo, and NSIS.ay.

40 http://netweb.ing.unibs.it/~ntw/tools/gt/
41 http://netweb.ing.unibs.it/~ntw/tools/traces/
42 https://csr.lanl.gov/data/2017.html
43 https://www.unb.ca/cic/datasets/ids-2017.html

- The Indian River State College (IRSC) data set is a data set from 2015 where real traffic recordings were run through the Snort IDS to label the attacks. The data set reportedly consists of a flow data set (IPFIX and NetFlow) and a packet data set [343].

- UGR'16 is a NetFlow data set in an ISP network [201]. The data set features real background traffic and generated, labeled attacks. Low-rate Denial-of-Service SYN flooding attacks were conducted with hping3, network scans (SYN scans) with Nmap. Neris botnet traffic was added from the CTU-13 data set. The attack runs are documented in detail. The data set is available online[44].

- The Information Security and Object Technology (ISOT) lab at the University of Victoria provides several data sets for numerous use cases[45]. In the network realm, two botnet data sets are of interest, the ISOT botnet data set and the ISOT HTTP botnet data set. The ISOT botnet data set is a combination of several publicly available data sets (LBNL and from the French chapter of the Honeypot project). The data is meant for the detection of P2P botnets [266]. The ISOT HTTP botnet data set contains malicious DNS traffic generated by different botnets and benign DNS traffic in a separate file [48].

2.1.4 *Special Purpose Data Sets*

For special cases — such as specialized infrastructures, for example, industrial control systems or car to car communication — the aforementioned more generalized data sets cannot be used as, for example, they do not represent the operations network of the device under test. Specialized data sets are necessary for those use cases. Some examples of these are listed below.

- The data sets created by Hofstede et al. to test their SSH attack detection tool SSHCure [147] consists exclusively of SSH network traffic. They contain host-based log files that can be used to assess the flow-based network recordings to check if a login attempt was successful. The data sets are available on Simple Web[46].

- The UMass data repository[47] run by the University of Massachusetts Amherst Laboratory for Advanced Software Systems provides data sets from the Laboratory and from third parties for a large variety of applications such as a data set of *Cellular*

44 https://nesg.ugr.es/nesg-ugr16/
45 https://www.uvic.ca/engineering/ece/isot/datasets/
46 https://www.simpleweb.org/wiki/index.php
47 https://traces.cs.umass.edu/

Phone GPS, Signal Strength, and TCP Data. The data sets that can be found in this repository are meant for specific use cases, neither general network traffic of ISP networks nor Denial-of-Service attack traffic can be found here.

- The AWID project [178] from 2015[48] consists of two data sets of different sizes from different networks and different locations. Both were recorded in wireless networks with typical home network devices such as smart-TVs and laptops as client machines. The two data sets can be downloaded with two different kinds of labels for the attacks. In one case, every attack has its own label, in the second case, three different attack classes (flooding, impersonation, and injection) are grouped together. The sets are available in CSV file format. While the comprehensive version contains 37 million and 4.5 million records, the reduced version is comprised of 1.7 million and 500 000 records. Access to the data set is given by request only.

- The power system data set (PSD)[49] of 2015 from the Oak Ridge National Laboratories is focusing on industrial control systems (ICS) split in three parts: a power system data set, a gas pipeline data set, and a gas pipeline and water storage tank data set [238–240]. In addition to standard tasks of such systems, some simple attacks and exceptional events (e. g., power decrease because of natural phenomenons) are included. Due to the focus on ICS, PSD only contains a limited, specialized set of attacks such as the infusion of falsified sensor data and malicious modifications of system parameters. However, some of the included attacks are only possible if conventional attacks occurred in advance. The set is available in the CSV file format that also includes sensor data of various sensors within the ICS. Statistics were collected with the help of Snort.

- The Cloud Intrusion Detection Dataset (CIDD)[50] from the University of Pisa extended the DARPA data set with a masquerading attack in cloud environments. CIDD was built based on host data (logs) and network data (records), making it usable for both HIDS and NIDS analysis. Unfortunately, nearly no information about this data set is available except for the main publication [172] and the project's website. Only a few record-samples are available, which include very few attacks, labeled for the specific records, and which let us assume that the records consist of recordings conducted over several weeks. In addition, the authors mentioned that more attacks such as malware and few others are included within the main data set.

48 https://icsdweb.aegean.gr/awid/download.html
49 https://sites.google.com/a/uah.edu/tommy-morris-uah/ics-data-sets
50 https://www.di.unipi.it/~hkholidy/projects/cidd/

For further reading on available data sets and a different classification and insights into the data sets, we would like to refer to the survey by Ring et al. [262].

2.1.5 *Conclusion*

Data sets are a vital instrument for network testing. However, as can be seen, there are limitations. Recordings of real traffic usually do not provide information on attacks present in the network (i. e., the data is not labeled), and anonymization can limit their application areas. Many recordings are comparatively old, which limits their usefulness in current applications. For many applications, researchers use simulated traffic as a way to circumvent these limitations.

2.2 TRAFFIC MODEL ANALYSIS

Leonard Bradatsch [17] has contributed to this section with his master thesis.

With the limitations of recorded data sets in production networks, for some analyses, other possibilities have to be found. For example, protocols that are not yet found in production networks cannot be found in recorded data sets of production networks. Live simulation of traffic in a software-based or hardware-based test environment is one way to ensure flexibility for analyses that cannot be realized with live production networks or with recorded data sets. However, it is no easy task to assure that the simulated traffic is comparable to real recordings. One approach is to find the most important and defining metrics of recorded network traffic and then, in a second, step simulate that exact behavior as closely as possible. As our goal is to emulate Internet traffic, prior to this, we need to identify distinct traffic models as the Internet is a mixture of many different types of traffic.

Cáceres et al. [89] assessed the characteristics of wide-area TCP traffic and described a way to model the wide-area traffic. The model is meant to be used to study congestion control, routing algorithms, and resource management schemes both for existing and future networks. The authors recorded Internet traffic at iconectiv (*Bellcore*) as well as at the University of Southern California and the University of California, Berkeley and analyzed its core characteristics. They reported the cumulative probabilities of conversation durations and the conversation packet counts, packet sizes, interarrival times, as well as the probability number of concurrent conversations, or the conversations per hour.

A survey on traffic models by Chandrasekaran [96] gives an in-depth view of network traffic characteristics split up into the most common traffic classes. The survey contains a description of mathematical models useful for the documentation of network traffic patterns. The author discusses the Poisson distribution, Pareto distribu-

tion, and Weibull distribution and how they can be used to model network traffic classes.

The *Handbook of Computer Networks* by Thomas M. Chen [81] contains a large variety of statistical models commonly used to model network traffic types. Rudimentary models for application web traffic, peer-to-peer traffic, and video streaming are part of this work.

Furthermore, specific traffic types are often analyzed separately. Staehle et al. [290] analyzed web traffic, FTP, and wap traffic and build models for web and e-mail traffic. Vicari and Koehler [311] analyzed user behavior faced with different access bandwidths and latencies in the network.

There are also dedicated analyses for web traffic [78, 98, 252, 315], file sharing [42, 58, 82, 85, 124, 136, 185, 292, 330], storage and marketplace traffic [173], and buffered video entertainment [62, 183, 256] These analyses contain enough information to produce realistic data sets for the different traffic classes.

2.3 EVALUATION PROGRAMS

Many programs can help in the evaluation process. Here, we focus on the programs relevant to our use cases. For a more comprehensive survey in the area of traffic generators, we would like to refer to Molnár et al. [223].

To use the aforementioned data sets, programs are used to replay the data sets into the network. For replaying pre-recorded network traces, Tcpreplay[51] or GoReplay[52] can be used. In addition to replaying recorded traffic, Tcpreplay also offers editing features for Pcap files. While Tcpreplay focuses on the lower layer of replaying traffic, GoReplay focuses on the capturing and replaying of HTTP traffic only.

Both benign and attack traffic is necessary to test security applications. The replay programs can be used for both. However, in some cases, when replaying data sets is not sufficient, traffic generators are used. They typically only produce benign traffic. Ostinato[53], packETH[54], and iperf3[55] can build and subsequently send user-customized or randomly generated packets. They support most established protocols on the lower Internet layers, for instance, IPv4 and IPv6, TCP, and UDP. The payload is usually randomly generated data. The FLExible Network Tester (Flent)[56] builds on some of these tools specifically to automate test runs and their analyses. D-ITG [66] is a traffic generator that consists of a client and

51 https://tcpreplay.appneta.com/
52 https://goreplay.org
53 https://ostinato.org/
54 http://packeth.sourceforge.net/packeth/Home.html
55 https://iperf.fr/
56 https://flent.org/

a receiving-only server side. This open-source tool can simulate the client-to-server traffic patterns of various applications such as Telnet, DNS, or Quake3. The flow characteristics are based upon common stochastic processes for packet inter-departure time and packet sizes. In addition, both parameters can be adjusted by the user. D-ITG does not include functionalities such as bi-directionality of the flows, adjustable file object sizes, or concatenated distribution models. Other existing tools focus on specific traffic types, mostly web traffic. The toolset Mahimahi [234] is capable of — among other things — recording and replaying HTTP traffic. The replaying function reproduces the recorded packets between virtual hosts running on one computer. The Web Traffic Generator[57] generates web browsing traffic by visiting real web pages.

Most of these tools do not provide bi-directional traffic. In most cases, traffic is sent at the highest rate possible, which is usually not realistic behavior of real network clients. MoonGen is a *"high-speed traffic generator"* [116]. Settings can be changed through the configuration of Lua scripts; the traffic is then generated with DPDK. MoonGen makes use of PF_RING Zero Copy to achieve higher throughput. In contrast to many traffic generators where traffic is usually just sent at the highest rate possible, MoonGen features rate control based on the Poisson distribution. To achieve this control over the sending rate, gaps between the packets need to be induced. One way is to delay packets in software. The authors of MoonGen were not satisfied with the results of such software-controlled measures and instead induced faulty packets (i. e., wrong CRC checksums) as gaps. The drawback of this method is that the device under test (DUT) needs to discard these faulty packets correctly and that the discarding of packets shall in no way affect the test results, which is hard to guarantee. Furthermore, measuring the traffic rate between MoonGen and DUT can be affected by this.

There are some generators that induce artificial attack traffic into the network traces. Flame[58] induces malicious traffic into benign traces before sending [70]. As Flame is limited to induce attacks into network flows, Cordero et al. [102, 308] decided to implement ID2T[59] that is able to inject exploits into the network packets. GENESIDS uses Snort-like rulesets to create strings that trigger these very rules and introduce them into the traffic[60] [117].

Often, these traffic generators are embedded in network simulators, especially when no real network hardware is available. NetSim[61] and ns-3[62] are simulators for network environments including

57 https://github.com/marty90/WebTrafficGenerator/blob/master/Readme.md
58 http://www.flame.ee.ethz.ch/download.html
59 https://github.com/tklab-tud/ID2T
60 https://github.com/felixe/idsEventGenerator
61 http://netsim.org
62 http://nsnam.org

traffic flows. Mininet[63] is a network simulator that can be used to simulate some aspects of the physical layer such as bandwidth, congestion, packet drops, and latency. OMNeT++[64] is a discrete event simulator that is used to build large scale event-driven environments. *Simulation of Urban MObility (SUMO)* can be often seen in automotive research to simulate traffic environments of whole cities but can also be used for network simulations. The focus of these systems is to look into large scale effects of networking decisions. Lack of full physical layer simulation and realistic traffic modeling within the simulators means that they need additional programs taking care of these parts of the simulation when necessary.

Even when the hardware is not simulated, frameworks that can control certain aspects of the networking environment and automate testing can help achieve better results faster. These systems are more research-focused developments. Dumitrescu et al. developed the distributed performance-testing framework DiPerF [114]. The framework consists of a controller instance that controls several nodes in the network. They can send unidirectional traffic from one node to another and can be used to stress test services or devices located between the nodes. GridBench [321] by Tsouloupas and Dikaiakos is a tool focused on the benchmarking of grids and grid resources. The system contains mechanisms for collecting, archiving, and publishing results. Neither of these tools is able to produce bi-directional traffic or is able to simulate physical network features.

Some programs and hardware implementations can be used to emulate the typical behavior of WAN environments. NetEm [299] is a program that can emulate features of wide-area networks in a receiving host. It supports packet drops, duplication, loss, and re-ordering of packets. Similarly, the WanRaptor Network Emulator[65] is a hardware box that can emulate bandwidth, latency, loss, and jitter of wide-area networks. The Spirent TestCenter[66] is a set of commercial tools to facilitate network tests. The system contains both hardware and software implementations to simulate network behavior.

These programs can be used to produce benign traffic. Penetration testing tools are necessary to produce malicious traffic. Kali Linux is a Linux distribution specifically optimized for penetration testing. It comes with a large variety of programs installed that can be used to run any type of network-based attack. The programs explained in the following are all also available in Kali Linux. Metasploit is an open-source framework for penetration testing[67] that offers automatic attacks such as trojan horses and brute-force attacks. It can be used for many network-based attacks. However, it does not provide

63 http://mininet.org
64 https://omnetpp.org
65 https://ecdata.com/wanraptor-network-emulator.html
66 https://www.spirent.com/products/testcenter/platforms/software
67 https://metasploit.com

options for Denial-of-Service attacks. For flooding DDoS attacks of any kind, hping3 is usually used[68]. With this program, when enough virtual interfaces are added to the Linux kernel (and the network permits it), a DDoS attack can be launched from only one client, given that the client's hardware is strong enough. For low and slow Denial-of-Service attacks, other tools need to be used. For the slow READ attack, Slowloris[69] offers the most common implementation. The slow POST attack is implemented, for example, in R-U-Dead-Yet (RUDY)[70]. SlowHTTPTest offers an implementation of both attacks. The IDS stimulator Mucus [229] was developed to automate testing of Intrusion Detection Systems. The tool comprises a parser and a traffic generator. It uses the Snort community database of attack signatures to build payload data that in turn triggers these rules. The authors conducted tests of Snort and the Net Prowler IDS with Mucus and real attacks they conducted themselves and concluded that in nearly all cases, the simulated Mucus traffic and the attack runs produced the same results at the IDS.

All in all, evaluation programs are plentiful. However, they are mostly specialized for specific use cases or traffic classes. While a system that can produce a realistic mix of traffic classes for network testing is missing, the approaches discussed here can serve as a basis for such a system.

2.4 TESTING METHODOLOGIES

Besides data set and evaluation programs, some additional aspects need to be considered to allow us to perform network tests. While until this point, we looked into general requirements for network testing, in the following, we focus more on specific issues in the network testing areas relevant for this thesis. Looking into the high-throughput aspect of network testing, we focus on the area of TCP congestion control evaluations as a typical example of protocol testing. We look into IDS testing for the security aspect of network testing, as this will be important for the following parts of this thesis.

2.4.1 *IDS Testing*

Puketzka et al. [253] describe in detail how the procedure of testing an IDS should look like. Although they look at testing host-based IDS, their approach is also applicable to network-based IDS (NIDS). They distinguish three different test scenarios: intrusion identification test, resource usage test, and stress tests.

Christian Forst [20] has contributed to this section with his master thesis.

68 https://tools.kali.org/information-gathering/hping3
69 https://github.com/gkbrk/slowloris
70 https://sourceforge.net/projects/r-u-dead-yet/

In the intrusion identification test, the IDS is faced with benign and attack traffic and has to find and identify the attack within the traffic correctly. For NIDS, it is imperative first to define the successful detection of an attack. One possibility could be to define an attack as detected if any part of the attack is detected. For example, for an SSH brute-force attack, the attack would be successfully detected if at least one of the login attempts is correctly identified as an attack. Another option would be to measure any packet or flow belonging to an attack separately. Furthermore, this definition then has to be applied to benign traffic to find a suitable metric for true negatives and false positives. It is imperative to define this clearly for the data set used for the analysis before running the first test.

The resource usage tests analyze which resources are used and how extensively they are used. Resources that should always be analyzed are memory and CPU usage. Additionally — depending on application — bandwidth utilization is important to measure. If applicable, bus utilization and GPU utilization should also be reported. Measuring these resources then, in turn, helps to find bottlenecks and explain performance issues.

The stress tests analyze how IDS perform under high load scenarios and try to determine the maximum load under which the IDS still performs correctly. These tests can consider spikes, prolonged high rates, or large variations of loads.

Choosing the right data set or data sets for the evaluation can also lead to very different results. Many features need to be considered when choosing a data set, which will be explained in the following.

- One of the main features of a network that needs to be considered is the *amount of participating devices* or the number of nodes. Depending on the scenario, this number can vary greatly. One example of a small network is the Small Office / Home Office (SOHO) network, with just a few devices. Networks of medium size are, for example, the networks of companies or universities. Big networks are, for example, ISP backbone networks. Rough estimations of network sizes we will use in the following could look like this: A low amount of network participants with less than 100 devices, a medium network size with 100 to less than 1 000 participants, a high amount with 1 000 devices or more but less than 50 000 devices and a category with a very high number of more than 50 000 devices.

- Another feature is the typical traffic load or *average bandwidth utilization* that is to be expected. A device under test (DUT), for instance, an IDS, has to be able to scan the number of packets of the field installation network without packet drops. Therefore, a data set used to test the device needs to facilitate a comparable amount of traffic. Taking into account observed networks

and data sets, we suggest a division of four data rate groups of below 100 Mbps, 100 Mbps to under 10 Gbps, 10 Gbps to under 100 Gbps and 100 Gbps and beyond as low data rate, medium data rate, high data rate, and very high data rate respectively.

- Data rates in networks usually do not remain smooth over time. In many networks, it is to be expected that *load peaks* happen either regularly or at random. Load peaks bear the possibility to cause packet drops and buffer overflows when the DUT is not designed to manage those. Therefore, we classify networks in three categories with no expected load peaks, irregular load peaks, and regular load peaks. In the case of regular load peaks, in addition, the frequency should be noted.

- Another aspect is the *variability* of the network topology. The network variability can manifest itself both in changes of the routing within the network and in changes of the position of devices in the network accompanied by fluctuating reachability of devices. Networks can be classified into three groups. In networks with high variability, changes in the topology can be observed at least daily but might also contain more frequent changes. An example of such a network could be a network with WiFi Access Point. In networks with medium variability, topology changes are to be expected but do not frequently happen. These include typical stationary home and office networks or data centers with cloud instances that are started or stopped on demand. Networks with low variability typically do not experience topology changes or are rare enough to neglect them when evaluating a productive network. An example of such a network could be some data centers.

- These four features cover a big part but not all distinctive features of different network scenarios. Some networks do feature additional, special qualities (e. g., scenario specific protocols, for instance, in the automotive or IoT context) that are to be considered individually.

How these features change depending on network type are described in the following.

- A *Data Center* can differ greatly in size and can reach up to 10 000 communication partners in the network [60]. The topology may be very diverse. For instance, the network can feature a star topology or a three-tier data center network topology. However, network topology typically does not change often. The protocols and applications in the network usually comprise authentication protocols and services such as LDAP, HTTP, and HTTPS-based web applications. They might feature for the operator specifically developed programs and services. Therefore,

in addition to standard web protocols, proprietary protocols, and protocols specific to data centers (e. g., map reduce protocols) can be found. These factors are very specific for each data center. Load peak occurrence depends on the link under surveillance. Edge links and aggregation links only show a low variance of 10% on average, while core links show even lower peaks [60].

- Conventional—e. g., *company networks*—show a large amount of IP-based protocols. E-Mail (SMTP, POP3, IMAP), HTTP, VoIP, Telnet, SSH, and similar protocols can be observed in these networks. There are high daily fluctuations with low traffic at night, high bandwidth utilization during work hours during the week, and low bandwidth utilization during weekends and holidays.

- *Internet backbone network traffic* typically consists mainly of IP packets with additional backbone routing protocols such as the Border Gateway Protocol (BGP) that, while highly relevant for network operations, make up only a small amount of traffic. An evaluation of 7.6 TB of Internet backbone traffic showed that only 0.03% of all frames use those routing protocols [160]. IDS in backbone networks need very high throughput. Central nodes, such as the Frankfurt-based DE-CIX, can have peak throughputs of 6.8 Tbps [235]. Peaks can usually be observed between 8 pm and 9 pm and correlate strongly with the sleep-wake rhythm of the population [324]. The number of devices in the network can reach more than 100 000 simultaneous users. The mixture of protocols, applications, and other variables in the network is very diverse. Packets of the same protocol but from different senders can differ greatly because of differences in the implementations or the usage of non-standard protocol headers. Therefore, homogeneity in the network is very low. The structure of a backbone network seldomly changes, which justifies the network's classification as a low dynamic network.

- *Industrial control systems* are predominantly in use to control physical components in the field. There are special requirements for this network type. Examples include, real-time reactivity, low bandwidth requirements, specialized network equipment, focus on reliability, and special protocols. Some standard protocols such as FTP can be found seldomly. Instead, protocols such as PROFIBUS, FIP, or CAN can be encountered frequently [125]. The concrete composition of protocols is highly dependent on the actual network components in use. However, the network composition tends to stay quite static over time [208]. Analyses of industrial control networks show traffic rates between 8 and 11 Mbps with 600 to 800 concurrent

connections [208]. The bandwidth utilization stays comparably constant and predictable because it is mainly characterized by sensor updates of field equipment, instructions by processing units, and status update requests by monitoring systems that tend to be scheduled, frequent processes. However, fluctuations can occur when alarms are triggered.

- *Wireless networks* display a high dynamic and fluctuating reachability of network devices. Endpoints can change their position in the network frequently while the connection to other systems should still be maintained. Except for the endpoints, the network topology stays broadly constant. Wireless LAN networks tend to show a similar composition of network packets as private or company networks but also show additional, IEEE 802.11 typical extensions.

2.4.2 *Protocol Testing*

Leonard Bradatsch [15] has contributed to this section with his bachelor thesis.

Parts of this section have been published at IEEE LCN 2016 [1, 2]

In the area of protocol testing, the first step is to analyze the goals of the protocol. This includes the handling of unwanted but common circumstances (e. g., packet drops). Critical in that regard are recovery times and reliability. Resources should be used efficiently, i. e., should be fully utilized when necessary. In some cases — for example in case of TCP congestion control — another central goal is fairness between network participants, i. e., fair distribution of network resources. For distributed systems, this fair distribution often takes time — the convergence time of the protocol. As a common example of protocol testing, we choose TCP congestion control. In many TCP performance evaluations [138, 323], the criteria most frequently used to evaluate TCP congestion control algorithms are the ones described by Li et al. [195] explained in the following.

Responsiveness describes the ability of the algorithms to recover quickly from random packet loss. For this, the average throughput at different drop probabilities for a packet in the network must be measured. This is also influenced by different propagation delays as this has a major impact on the recovery speed.

Efficiency is the utilization of the network resources, i. e., the share of the available resources that are utilized. A protocol is efficient if the available bandwidth at the bottleneck is utilized as fully as possible. In some applications — in addition to this metric — not only the average utilization is important but also the maximum and minimum utilization. For instance, when incoming data is processed immediately, applications will be slowed down if the bandwidth fluctuates heavily. Therefore, it is also beneficial to assess the quantiles to measure if high throughput at its peaks can be accomplished (Q 0.75) if the average throughput is acceptable (Q 0.5) and if the throughput has an acceptable minimum (Q 0.25). In addition to the responsive-

ness, which shows the average throughput, this metric shows how stable the algorithms are.

Fairness is usually measured with Jain's fairness index [156]:

$$J(x_1, x_2, \ldots, x_n) = \frac{(\sum_{i=1}^{n} x_i)^2}{n \cdot \sum_{i=1}^{n} x_i^2} \tag{1}$$

with x_i being the mean of the throughput of flow i with n flows overall. A perfect algorithm would result in $J = 1$; the worst case would be $J = \frac{1}{n}$.

Backwards compatibility is a measurement of fairness within networks where older systems are still in use. The level of fairness in heterogeneous TCP networks estimates the backward compatibility in legacy networks. It can be measured by using older protocol implementations for one part of the network flows and the new edition for the other part and reapplying the aforementioned fairness metric.

ε-Convergence time t_c was defined by Li et al. [195] as the time required for the short-term average throughput to achieve $\varepsilon \cdot \bar{u}_i$, where $\bar{u}_i$ is the long-term average throughput of stream i. Here, the short-term average throughput is defined as:

$$u_i(t + \delta) = (1 - \lambda) \cdot u_i(t) + \lambda \frac{\Delta u}{\delta} \tag{2}$$

Where Δu is the number of bytes transferred, and λ is a parameter that specifies how quickly the short-term average throughput changes. Results from [195] suggest that convergence time should be measured by analyzing this newer stream as opposed to the stream already active in the network.

$$s = \frac{\sum_{t=t_c}^{T} |\bar{u}_i - u_i(t)|}{T} \tag{3}$$

There are many TCP performance evaluation papers that cover environments with link speeds up to 1 Gbit/s [138, 323], but there are very few that look at 10 Gbit/s transmission rates. As we focus on high-bandwidth network testing, in the following, we focus on relevant studies from this body of work.

Li et al. [195] measured the performance of Scalable TCP, HS-TCP, H-TCP, BIC, and FAST-TCP on the basis of fairness, backward compatibility, efficiency, and responsiveness, including convergence time. All tests were performed in a test setup based on the dumbbell topology with two competing flows starting at different points. The authors varied the parameters of propagation delay (up to 320 ms), the bottleneck bandwidth (up to 250 Mbps), and different numbers of parallel web traffic flows. The TCP variants under test provide poor fairness but better link utilization than standard TCP. Beyond that,

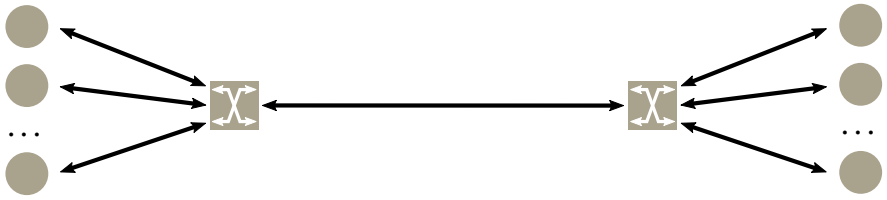


Figure 2: Dumbbell topology [2].

the algorithms Scalable TCP, HS-TCP, and BIC suffer from high convergence times.

Arokkiam et al. [329] evaluated the performance of the TCP variants Reno, BIC, and H-TCP over XG-PON. The authors assess the results on the basis of efficiency, fairness, responsiveness, and convergence. One single or two competing high-throughput flows were induced, alternately with or without competing UDP background traffic, into a 10 Gbps XG-PON network. All algorithms show good link utilization in a single flow environment with very small RTTs, but the link utilization decreases with increasing round-trip times. This paper provides an extensive analysis of the mentioned TCP variants. However, key variants such as CUBIC and HS-TCP are missing.

Hock et al. [145] analyzed BBR, the new congestion control mechanism presented by Google. Their experiments vary the round-trip times of flows, the number of flows, and the buffer sizes at the bottleneck. The evaluation considers effective throughput, queuing delay, packet loss due to congestion, and fairness.

2.5 TOPOLOGIES

The standard test setup for many performance tests is the dumbbell topology (Figure 2). The topology consists of the same number of sender and receiver nodes. All sender nodes are connected to a switch, which forwards the incoming flows over one link to another switch, where they are again separated and forwarded to the receiver nodes. The link between the two switches constitutes a bottleneck, which leads to congestion and, therefore, for example, to the observability of congestion control algorithms. Furthermore, a device under test can be located here. The aforementioned criteria for protocol testing need different numbers of flows in the network. To evaluate responsiveness and efficiency, only one sender and receiver are active. When assessing the other metrics, two or more sender and receiver nodes are necessary. Besides the dumbbell topology, there are several other possible topologies, such as the parking lot (Figure 3) or the ring topology (Figure 4). While dumbbell is sufficient for most evaluations, other topologies can help to examine very specific network configurations. The parking lot topology can be used to simulate multiple bottleneck links, for example, a server close to the backbone and clients with different distances to the server. The ring topology is

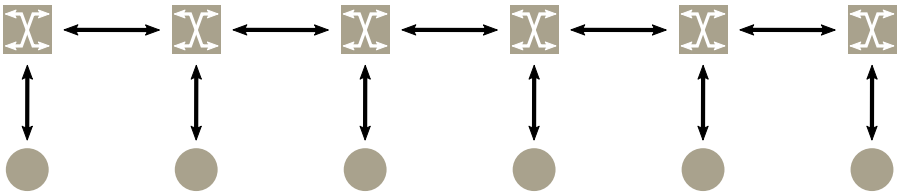


Figure 3: Parking lot topology [2].

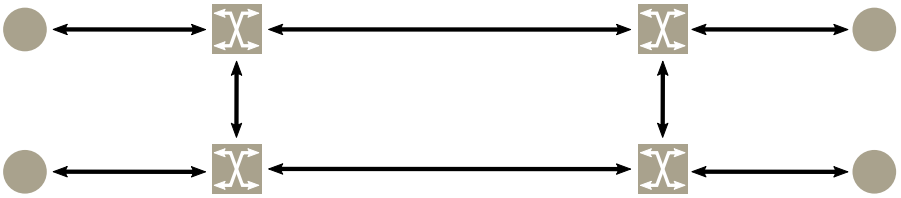


Figure 4: Ring topology [2].

used to observe protocols with asymmetric routing. We would like to refer to Wei et al. [321] for further reading. Wei et al. give a detailed overview of the typical topologies extending on the listed topologies here and when each of those should be used.

The specific test configuration depends on the use case. Traffic patterns can differ greatly depending on the observed network. For example, one short request to a web server can lead to subsequent requests to other back-end (e. g., database) servers. These traffic bursts have to be simulated differently than constant flows that can be observed in ISPs' backbone networks. Therefore, some criteria such as best fitting network topology, the number of flows concurrently in the network, which protocols should be observed, and if the sender and receiver nodes should use the same implementations should be considered carefully. Additional background noise that emulates the typical environment of the specific use case can also be an option. The aforementioned data sets and programs can be used to induce background traffic. The protocol mix of the use case network has to be taken into account, i. e., the ratio of UDP to TCP or other protocols in the network, how different devices interact, or if the traffic is equal in both directions or tends to be more unidirectional.

2.6 SUMMARY

Network testing requires considerations in several different areas. Programs that can replay or produce data exist but are often very application-specific. Testing topologies are often limited in scope but can accurately represent certain aspects necessary in the concrete testing scenario. Data that accurately represents the target network needs to be considered carefully as not all data sets can be used for all use cases. Using recorded data sets allows us to perform realistic tests. In turn, simulated traffic allows for higher adaptability to future trends and extreme outliers.

3

PROBLEM STATEMENT

Devices, services, protocols, and other network mechanisms require evaluation before they can be used in production. An excellent test infrastructure for a new network system is the production network in which it has to function. However, tests in such infrastructure have their downsides. For one, it is hard to gain access to these systems as usually privacy regulations, and policies understandably forbid the usage of unproven and untested devices in the production network. Moreover, the infrastructure cannot be controlled, which leads to a lack of repeatability and reliability of network tests. The state of the art to work around this issue is to record, anonymize, and reuse network traces and — at least as a best practice — publish these data sets along with the results of the analysis. As they are recordings of real network traffic, they can assure that they accurately reflect features of the network traffic that even the tester might not have thought of. The inherent limitation in this approach is that these network traces only depict the properties of networks during their recording time. Traffic patterns could possibly only be present within the network under observation, maybe even only at the time of recording. Newer protocols or changes in user behavior lead to fast obsolescence of these network traces. For instance, HTTP/2 or the QUIC protocol cannot be found in older data sets. As it is hard to gain access to newer traces or to ensure comparability with older tests, systems are often tested against old data sets that do not represent current network features, for instance, the DARPA Intrusion Detection Evaluation data set from 1999 is still in use today. Another critical point of tests is to not only test in a realistic environment but also to test edge cases and push the network system to its limits. This cannot easily be done in a production network or with traffic recordings. Therefore, specialized systems for network tests that can produce a realistic and controllable environment are important to prove the viability of new network systems. These systems should not only be able to reuse existing traffic traces but should also be able to produce new data.

We lay our focus here on two aspects of network analysis. The first aspect is the high throughput necessary to test high-bandwidth network applications. To analyze this aspect, we take a look at network protocols, and, more precisely at TCP congestion control algorithms as a common use case. While there are many evaluations analyzing TCP congestion control algorithms at lower bandwidth that serve us as a reference point, we tackle the lack of evaluations of these algorithms in high-bandwidth networks in the literature in Chapter 4.

The second aspect of network analysis is the testing of security applications such as Intrusion Detection Systems and also more precisely, the testing of DDoS mitigation system as a preparation of the following parts III and IV. For this purpose, we built the General Purpose Network Testing Framework introduced in Chapter 5 that combines high throughput and testing of security applications.

3.1 RESEARCH QUESTIONS

In the following chapters, we want to answer the following research questions:

1. Network testing in high-bandwidth networks: Evaluation of TCP congestion control mechanisms in 10G networks as a use case.
2. How can benign traffic be modeled and produced to test network mechanisms?
3. How can malicious traffic be modeled and produced to test security network mechanisms?

In order to obtain the necessary results to answer these questions, the following steps are necessary:

- Setting up a testing environment including WAN portions for the protocol tests.
- Building a framework — the General Purpose Network Testing Framework — that can produce both realistic live traffic and data sets.

4

USE CASE: EVALUATION OF TCP CONGESTION CONTROL ALGORITHMS IN 10G NETWORKS

This Chapter is based on the combination of two previous publications at IEEE LCN 2016 [1, 2].

Leonard Bradatsch contributed to this Chapter with his bachelor thesis [15].

As we need to test the performance of security mechanisms in high-bandwidth networks, first, we need to make sure that we can achieve the throughput required. One of the main building blocks of networks are the network protocols that define the network behavior on different levels. Influenced by our work in the research project bwNET100G+[1] focused on the requirements of the Research Network Baden-Württemberg BelWü [2], we found the behavior of TCP congestion control in high-bandwidth networks as a topic of interest and a fitting use case for a network testing setup. To give us a point of reference to begin this work in the area of network testing, we choose this common area as a starting point.

The concept of layering separates concerns on different levels. A very prominent example of this approach is networking and, in particular, the Internet network protocol stack. This approach leads to the complete separation of tasks. However, different layers do influence each other in very obvious and sometimes less obvious ways. Higher layers build on the services of lower layers and depend on their capabilities. The transport layer in the Internet protocol stack is responsible for providing end-to-end communication services between remote applications. Congestion control is one of the common services of transport layer protocols. By limiting their sending rate, network participants can distributively and collaboratively optimize the network bandwidth utilization. Although located on the transport layer, congestion control relies on a set of properties of the underlying network links, primarily latencies, drop rates, error rates, and bandwidth utilization. The continuous increase of available bandwidth since the inception of the Internet has a significant effect on the performance of congestion control algorithms. When the increasing link bandwidth raises the bandwidth-delay product (BDP), the effects of occasional packet loss impact the utilization disproportionately and requires counter-measures by extending TCP [154].

Specific TCP congestion control algorithms were developed to address the challenges introduced by high-bandwidth networks. Through adapting behavior and optimizing window parameters, these TCP congestion control algorithms aim for better and faster utilization while still remaining fair to unmodified TCP connections. With the advent of widely available 10 Gbit/s Ethernet networks, the

1 https://bwnet100g.de
2 https://belwue.de

current state of the art of TCP comparisons primarily focusing on 1 Gbit/s [138, 323] becomes outdated. We limit our evaluation to 10 Gbit/s, as higher bandwidths of up to 100 Gbit/s for single flow applications are still in the experimental stage [146].

The choice of tunable testing parameters such as latency and drop rate behavior of the network, the patterns and workloads of network traffic, and the congestion control algorithms to be tested constitute the overall framework for test runs. Based on the algorithm properties to be evaluated, various metrics must then be measured during the tests. Furthermore, a structured analysis of results is necessary to assess the protocols after the test runs and to gain insights from the test data. As already observed by Wei et al. [321], an evaluation of TCP congestion control algorithms requires rigorous planning and strict testing procedures, which makes it a good example for assessing a testing framework.

For the evaluation of the performance of TCP congestion control algorithms in a 10 Gbit/s Ethernet network, we discovered a number of additional testing challenges stemming from the higher bandwidth. As not all parts of the network stack scale equally well to new bandwidths, 10 Gbit/s environments generally force the testers to put more emphasis on hardware components of the setup. For instance, prior tuning of network hardware and NICs [190] becomes mandatory in order to achieve full bandwidths.

4.1 TCP CONGESTION CONTROL ALGORITHMS

TCP congestion control basically throttles the sending rate of a network endpoint to use as many resources as possible while still being fair to other network participants. Fairness is achieved by avoiding congestion in the network in a collective way. When the client observes congestion or is under the impression that congestion is imminent, they decrease their sending rate. There are two main indicators of congestion that a client can measure: packet loss and delay. TCP Vegas is one example of a congestion control algorithm that reacts based on increasing packet delays [68]. The other indicator is the observation of packet loss, for example, in TCP Reno. Usually, the increase in delay happens earlier than packet losses. Therefore, algorithms that are primarily based on packet loss are more aggressive as they react slower to congestion. This leads to the incompatibility of the two mechanisms as packet loss based algorithms dominate delay-based algorithms. Therefore, congestion control algorithms usually used as the default by most operating systems are packet loss based.

In general, TCP congestion control algorithms follow the principle of additive increase and multiplicative decrease (AIMD), meaning they increase their sending rate slowly (additive increase) until they observe congestion. Then, they decrease their sending rate abruptly

(multiplicative decrease). AIMD inherently offers fairness, which makes it ideal for congestion control. The following provides a brief overview of widely known packet loss-based TCP congestion control algorithms, specifically TCP Reno, Scalable TCP, HSTCP, H-TCP, BIC, and CUBIC. TCP Reno was the default TCP algorithm of Windows XP (replaced by Compound TCP since Windows Vista, which is a hybrid between packet loss-based and round trip time-based mechanism) while BIC was the default in Linux from version 2.6.8 up to version 2.6.19 when CUBIC became the default. Here, we only provide the additive increase and multiplicative decrease (AIMD) parameters, which are of particular interest for our analysis. The AIMD behavior of packet loss-based algorithms can be described with an additive parameter (ACK received) and a multiplicative parameter (triple duplicate ACK, packet lost):

$$\textbf{ACK} : cwnd \leftarrow cwnd + \alpha \quad \textbf{LOSS} : cwnd \leftarrow cwnd \cdot \beta \tag{4}$$

with the congestion window cwnd.

TCP RENO TCP Reno [123, 209] is also referred to as standard TCP and is one of the oldest congestion control algorithms. This variant initially uses slow start. At the start of the flow, the *slow start phase*, the congestion window increases by one with each ACK until a threshold *ssthresh* is reached. As the ACKs multiply by two in every step, the congestion window grows exponentially. After reaching *ssthresh*, additive increase starts with $\alpha = 1$. After a loss, cwnd is halved ($\beta = \frac{1}{2}$) and additive increase with $\alpha = 1$ continues from this point on. The following TCP congestion control mechanisms fall back to behave like TCP Reno for small congestion windows.

SCALABLE TCP Scalable TCP [169] is designed to achieve high throughput more quickly than Reno by making the recovery time independent of window size, which is beneficial for high bandwidth, high latency links. The AIMD parameters for Scalable TCP are $\alpha = 0.01$ and $\beta = 0.875$.

HIGHSPEED TCP HighSpeed TCP (HSTCP) [122] is designed to increase the robustness of the transmission rate against packet loss, which is especially important for networks with a large bandwidth-delay product (BDP). HSTCP uses the current TCP cwnd values as an indication of the BDP on a path. For this algorithm, the following AIMD parameters apply: $\alpha = f_\alpha(cwnd)/cwnd$ and $\beta = g_\beta(cwnd)$, where g_β (decreasing) and f_α (increasing) are logarithmic functions.

H-TCP H-TCP [191] is designed to provide a better use of bandwidth for long, high-bandwidth links with high BDP, while maintaining backward compatibility with regular TCP flows. Unlike pre-

vious approaches, the authors use the time (Δ) since the last congestion event to set the AIMD parameters, which can be summarized as follows: $\alpha = \frac{2(1-\beta)f_\alpha(\Delta)}{cwnd}$ and $\beta = \frac{RTT_{min}}{RTT_{max}}$, unless the measured throughput changes significantly (controlled with a parameter $\Delta_B = 0.2$). $f_\alpha(\Delta)$ is 1 for backward compatibility (below a threshold Δ_L); $f_\alpha(\Delta) = 1 + 10(\Delta - \Delta_L) + 0.25(\Delta - \Delta_L)^2$ is suggested to achieve high utilization quickly.

BIC TCP Binary increase congestion control (BIC) TCP [328] was developed to address the observed suboptimal round trip time (RTT) fairness of earlier congestion control algorithms. RTT fairness refers to fairness between flows with different RTTs. The authors point out that the problem is inherent to the increased utilization due to the way earlier algorithms are designed and develop BIC to solve this challenge. Their algorithm uses two phases to update the bandwidth; linear increase to approach a fair window size, and binary search to improve RTT fairness. Linear increase is similar to additive increase, while binary search essentially uses two window sizes (W_{max} and W_{min}) that updates these windows and the actual window size to approximate the optimal window size. Once W_{max} and W_{min} are converging, BIC falls back to linear increase.

TCP CUBIC CUBIC TCP [137] is an improvement of BIC, which aims to compensate for the aggressive behavior of BIC to more reasonable levels, and simplifies the algorithm. The impact of this aggressive behavior was especially notable in networks with low RTT. Similar to BIC, CUBIC uses the W_{max} window; however, it sets the window size using a cubic function that plateaus at W_{max}:

$$W(t) = C \cdot (t - K)^3 + W_{max} \tag{5}$$

where C is a scaling factor, t the time since the last window reduction and $K = \sqrt[3]{W_{max} \cdot \beta / C}$. This results in a window increase that is similar to BIC's binary search.

4.2 PLANNING A TCP BENCHMARKING ENVIRONMENT

Several decisions have to be made when building a test setup for TCP, such as choice of the network topology or which software tools to use. In addition, appropriate criteria have to be chosen to evaluate the properties of TCP. These points are crucial to attain a sensible test setup and test procedure.

4.2.1 *Criteria*

We already described the criteria usually used for TCP congestion control analysis in Chapter 2. However, we did alter them slightly.

- Responsiveness: We measure the average throughput at different drop probabilities for a packet in the network. We also measured this with different propagation delays, as this has a major impact on the recovery speed.
- Efficiency: In some applications — in addition to this metric — not only the average utilization is important but also the maximum and minimum utilization. For instance, when incoming data is processed immediately, applications will be slowed down if the bandwidth fluctuates heavily. Therefore, we chose also to assess the quantiles to measure if high throughput at its peaks can be accomplished (Q 0.75) if the median throughput is acceptable (Q 0.5) and if the throughput has an acceptable minimum (Q 0.25). In addition to the responsiveness, which shows the average throughput, this metric shows the algorithm's stability.
- Fairness: We analyze two flows running in parallel, each with the same configuration, for example, TCP variant and parameters.
- Backwards compatibility: We adapted our fairness test by using Reno for congestion control in one of the two flows to evaluate how the congestion control algorithms behave in networks with legacy systems. Reno was chosen as it is the most common legacy variant still in use.
- Convergence time: In practice, we observe that the newer TCP stream always takes longer to converge to its long-term average throughput; results from Li et al. [195] suggest that convergence time should be measured by analyzing this new stream. Extending the metric provided in prior work, we compute the average distance from the long-term average throughput after the convergence time is reached, in order to quantify the stability of this convergence, which we refer to as spread s (where T is the number of measurements):

$$s = \frac{\sum_{t=t_c}^{T} |\bar{u}_i - u_i(t)|}{T} \tag{6}$$

4.2.2 *Test Procedure*

HDDs are too slow to enable replay programs to send at high traffic rates such as 10 Gbit/s or higher and — depending on the specific hardware — this is also true for many SSDs. Even when the SSD is capable of this high throughput, read and write operations by the operating system or other processes can interfere with the read operations

and consequently affect the test results. Therefore, it is advisable to save the traffic recordings on a RAM disk.

Many improved versions of TCP congestion control algorithms were developed, following the now obsolete TCP Tahoe. As the usage of TCP Reno phases out, it still plays its role in many legacy networks and should still be considered as a baseline for comparison to the newer TCP variants. Another point to consider is the lack of availability of some algorithms for different operating systems. Compound TCP, for example, is only available on Windows. When comparing this variant with others, the tester is forced to use different operating systems for the comparison. Therefore, it can not be ensured that the difference in performance might not stem from a different OS or driver behavior. Another point to consider is that some algorithms are developed with downwards compatibility to Reno in mind while others do not work well with legacy TCP. Delay-based congestion control algorithms such as TCP Vegas might perform well in enclosed network environments but cannot be used when loss-based TCP variants are present [222]. A list of variants that explicitly aim for downwards compatibility includes Scalable TCP, HSTCP, BIC, CUBIC, and H-TCP.

Regarding network performance, the presence of flow control in switches represents another point to consider. For performance tests in networks, the question arises if flow control in switches should be turned on or off. Flow control means that switches send pause packets to senders if congestion is imminent. For the evaluation of an existing network environment, this setting should not be changed. However, it should be evaluated how differences in the client settings affect the performance of the network. For the dedicated evaluation of TCP congestion control algorithms, flow control should be turned off as it interferes with the congestion control and distorts the results.

To produce TCP flows with the desired bandwidth of 10 Gbit/s, *iperf3*[3] — a client-server-application to measure performance in networks — can be used. Version 3 additionally supports the detection of retransmissions during a connection. Additionally, iperf provides the possibility to adjust several TCP features such as the usage of the Nagle Algorithm (TCP no-delay) or the TCP maximum segment size.

4.2.3 *Kernel Settings*

Several options within the TCP specification can be tweaked to accomplish higher throughput. It depends on the operating system how the settings can be changed and to what extent this is possible. We concentrate on the features of the Linux kernel as it is both adaptable and wide-spread. The settings are changed using the *sysctl*

3 https://iperf.fr

command. The following settings have to be taken into consideration both on the client and the server side.

One very important setting concerns TCP Window Scaling. TCP offers the header field *window size* that determines the amount of data the receiver can collect without sending confirmation that the packet has been received. As a default, the window size field is 16 bit, which in turn results in a maximal window size of 65 535 bytes. According to Mathis et al. [210], the maximum possible transfer rate is the TCP window size divided by the round trip time. Therefore, to reach 10 Gbit/s with a window size of 65 535 bytes, a maximum RTT of 0.066 ms would be needed, which, even for local networks, is not a realistic value. RFC 7323 [65] offers a TCP Window Scaling Option to extend the window size. Both sender and receiver have to accept this option when establishing connections. An entry in the option field of the TCP header facilitates the multiplication of the window size field by up to 2^{14}. The window size can, therefore, reach up to 1 GiB. The limit in our evaluations is set in such a way that the maximum value cannot surpass the sequence number (saved in four bytes).

TCP Timestamp, as described in RFC 7323 [65], extends the precision of the timestamp options already present in lower layers to facilitate higher precision round trip time measurements. The TCP header is extended by 8 bytes when this option is turned on. In most cases, this option is not needed and should, therefore, be turned off as more capacity can be allocated to the payload.

In high-bandwidth networks, a high number of out-of-order segments can be observed. Due to the nature of the cumulative acknowledgments of TCP (i. e., Go-Back-N behavior), this leads to a high number of retransmissions. *TCP selective acknowledgment (TCP SACK)* [209] provides the possibility to acknowledge out-of-order segments explicitly and prevents a large number of retransmissions and should, therefore, be turned on.

The *Nagle Algorithm* [230] facilitates groupings of small data amounts to bigger segments and therefore leads to fewer packets in total, which in turn allows for higher bandwidth as the overhead of the TCP headers declines. In some cases, this can be obstructive when immediate feedback from the receiver side is required, for instance, in case of SSH connections. Nevertheless, Nagle is especially useful when many small packets can be observed and should be turned on if the use case permits.

Jumbo frames that allow for Ethernet payloads larger than 1500 bytes are advisable to facilitate high bandwidths. However, hardware support has to be examined as not all networking devices support jumbo frames. Therefore, if the test results should generally be applicable for networks where jumbo frame support cannot be guaranteed, a tester is advised not to activate jumbo frames.

4.2.4 *NIC Configurations*

Additional to the kernel settings, there are also important configuration parameters to consider for the NIC. *Large Receive Offload (LRO)* merges packets upon entry into a stack to handle large amounts of data more efficiently as fewer packets need to be processed, which also reduces the number of interrupts. As this introduces some drawbacks, notably being limited to IPv4, TCP, and unvirtualized environments, *Generic Receive Offload (GRO)* was developed, which fixes some of the issues. LRO and GRO lead to less computation overhead on the receiver end. However, in some driver implementations, if one of the merged packets was not received correctly, all packets need to be resent, which can lead to inferior link utilization due to the significant number of retransmissions. Furthermore, these mechanisms can affect the TCP congestion control behavior and consequently distort the results. Therefore, it is advisable to turn those features off.

4.2.5 *Network Adjustments*

There are — among others — two widely used adjustable network conditions to evaluate TCP performance: (1) packet loss and (2) packet delay. In a 10 Gbit/s testing environment, one can implement the aforementioned network conditions in different ways. There are several things to consider: Available hardware, evaluated uses cases, related metrics, and the tester's experience with different implementations accompanying the chosen approach. In the following, various approaches are described.

Packet Loss

There are various mechanisms to induce packet loss. The simplest but least predictable way is to expand the test topology by adding additional senders. These senders initialize additional network flows to create actual congestion on a bottleneck switch or router. It is extremely challenging to create accurate and repeatable packet loss using this approach.

Another way is to use a network environment emulator software. However, for a 10 Gbit/s test setup, the available software range is reduced to a few workable software tools. One of the best known network environment emulator is *NetEM*[4]. The user can adjust the drop chance and the burst length. The software needs to run on a computer with a 10G NIC with at least two interfaces to be able to forward the incoming packets. Both aforementioned methods facilitate constant or bursty packet loss.

4 https://wiki.linuxfoundation.org/networking/netem

Another possibility to induce packet loss is to use a hardware device in the network that usually forwards packets but can also drop packets at a specific rate. For example, the NetFPGA-Sume or its predecessor, the NetFPGA-10G[5], can be used as such a bump-in-the-wire forwarding device. The NetFPGAs are fully programmable network devices. A Virtex FPGA Processor on the PCIe NetFPGA card allows for traffic manipulation at line speed. With feasible effort, the official reference NIC implementation can be adjusted, and it is possible to implement packet loss with different drop patterns.

Packet Delay

Packet delay has a substantial impact on TCP sending performance as it — depending on the TCP congestion control algorithm — significantly influences the growth of the TCP congestion window. In addition to the physical propagation delay, transmission delay, packet processing, queuing time, or suboptimal routing can slow down a packet on its route.

As with packet loss, there are software tools available to emulate packet delay as it would occur in real networks. For example, *NetEM* is able to delay incoming packets for multiple milliseconds. It is highly recommended to check the available hardware, whether it is capable of buffering packets at a bandwidth of 10 Gbit/s and for how long this is possible. The available RAM — including data rate and read/write latencies — has a significant impact on the maximum possible buffer time.

In addition to software solutions, there are some hardware approaches. A 10G NetFPGA card can be used to simulate long-distance packet delay. The NetFPGA comes with RLDRAM, which provides low read and write latencies, and can buffer processed packets in a 10 Gbit/s environment. As part of the official NetFPGA repository on Github[6], there are some contributing projects, which can be used as a basis to implement a packet delaying device.

Hardware network emulators offer a more expensive but for the user less complicated approach. Such emulators are able to perform at line speeds of up to 40 Gbit/s. Additional features often include altering bandwidth, additional packet loss, or induced data corruption. The EDS-10/40G Ethernet Delay Emulator[7] can delay incoming packets for up to 8 seconds.

The solution closest to real network behavior — but also the most expensive one — is to use an actual long-distance network or a fiber cable drum with sufficient length. However, access to such a network or network equipment is typically not readily available, and long pipes often come along with quality impairments. Electrical

5 https://netfpga.org
6 https://github.com/NetFPGA/
7 https://ecdata.com/eds-10g.html

noise on the link can cause bit corruptions. Furthermore, transceivers and muxponders in the network can also cause bit flipping. These occurrences lead to CRC errors. TCP reacts with retransmission of the corrupted packets. Besides the induced packet loss and packet delay, these additional retransmissions have to be taken into account when analyzing the test results. The longer the round trip time, the stronger the impact of the additional packet drops on the TCP performance by causing extended recovery phases.

The performance evaluation of TCP variants in 10 Gbit/s high-bandwidth networks provides new challenges, as testers are required to take the network hardware into account even more closely. The choice of relevant testing criteria and appropriate metrics to be measured still governs the overall testing procedure. Responsiveness, efficiency, fairness, downwards compatibility, and convergence time are the primary characteristics for comparing different TCP variants. In order to vary the network environment, the test setup should allow the modification of different drop rates, data corruption, and end-to-end round trip times. We concluded that it is beneficial to induce such network characteristics using hardware-based solutions because software-based approaches so far have not matched our expectations regarding reliability and dependability in a 10G environment. In our setup, we relied on a NetFPGA to induce packet loss and corruption at line speed. Delays were induced using the propagation delay of a configurable network testbed with long-distance links.

In general, prior optimizations of the network stack are urgently required to be able to use full bandwidths in the tests. This includes, but is not limited to, network settings of the OS kernel, NIC configurations, and settings of network equipment used in the test topology, for example, advanced features of Ethernet switches.

If not considered carefully, one single subpar setting of the network setup can have huge effects on the actual performance of the TCP variants in a 10 Gbit/s network. With a change of bandwidth in an order of magnitude also comes a multiplying effect of any uncontrolled parameter confounding the results.

4.3 RESULTING TEST SETUP

Based on these deliberations, on available hardware capabilities, and the scenario for which the results should be applicable, we build our test setup as described in the following.

4.3.1 *Scenario*

We focus on high-bandwidth networks. Prominent examples of such networks are both data centers and WAN environments, such as intra-datacenter traffic. Alizadeh et al. [49] analyzed large amounts

of data center traffic and identified two major traffic types: for one relatively short flows with low latency requirements and large flows requiring high throughput. The first type is primarily a result of web application requests, database queries, and similar interactions within distributed application architectures. The latter type is based on long-running interactions, such as software updates, continuous database replications, data-intensive application workloads, or backup processes.

In the context of this work, we concentrate on the second traffic type, as it is more interesting to consider for high-bandwidth networks. Long-running, latency-insensitive flows run concurrently and compete for utilization on a shared high-bandwidth network. This leads to characteristics of different congestion control algorithms becoming apparent.

Furthermore, we include network traffic both inside and between data centers. The *intra-data center* traffic is characterized by shorter physical links and corresponding lower end-to-end latencies between nodes. The *inter-data center* traffic represents communication between geographically separated data centers, yielding much higher latencies. This use case includes the usage of multiple data centers for higher availability, increased locality, and improved resilience. The traffic patterns between sites produced by continuous data synchronization, database replication, and periodic backups remain very similar to traffic within a single data center site. Note that we expect data centers to have dedicated remote connections, as we do not take into account background Internet traffic for our tests.

Although we motivate our experiments with large flows between and within data centers, we believe that results can be generalized to many other use cases with similar traffic and network infrastructure properties.

Our test setup is based on the standard dumbbell topology with two senders and two receivers. Each of them is equipped with HP NC523SFP Dual 10 Gbit/s NICs and Ubuntu 14.04.1 (Linux 3.16). We used 10G Ethernet with IPv4 on the lower layers, as these are the most common protocols in our use cases. For the efficiency and responsiveness measurements, only one sender and one receiver were active. Two HP 5920 JG296A switches (Firmware HPE Comware Software, Version 7.1.045, Release 2422P01) were used to combine and separate the flows. Flow control in the switches was turned off. The program *iperf3* was used to produce TCP traffic with up to 10 Gbit/s for each sender. Several adjustments in the software settings were necessary to enable the senders to satisfy the bandwidth requirements. For example, *Large Receive Offload (LRO)* and *Generic Receive Offload (GRO)* had to be turned off. LRO and GRO merge packets in the NIC upon receiving them. As the loss of one packet means that the whole group has to be resent, this can lead to very low goodput. The *TCP Window*

Table 1: Used TCP parameters.

TCP variant	**Parameters**
Scalable	$\alpha = 0.02, \beta = 0.875,$ Low_Window = 50
HSTCP	Low_Window = 38, High_Window = 83000, High_P = 10^{-7}, High_Decrease = 0.1
BIC	$S_{max} = 16$, B = 4, $\beta = \frac{819}{2014}$, Low_Window = 14
CUBIC	$\beta = 717/1024$, legacy if *cwnd* $< W_{tcp}(t)$
H-TCP	$\Delta^L = 1s, \Delta_B = 0.2$

Scaling Option had to be turned on to allow bigger window sizes exceeding the default maximum of 65 535 bytes to a maximum of 1 GiB. The *TCP Timestamp*, which extends the TCP header by 8 bytes, was deactivated as it is not used in our scenarios, and the bandwidth can, therefore, be used for payload instead. *TCP selective acknowledgment (TCP SACK)* was activated to accommodate the high number of packets in 10 Gbit/s networks and to avoid retransmissions.

The *Nagle Algorithm* to facilitate groupings of small data amounts to bigger segments was also turned on to allow for higher throughput. Table 1 shows our settings for the TCP variants, and when they fall back to legacy mode, i. e., start to behave like Reno. These values are the default values of Linux 3.16 and are predominantly based on the parameters set by the original developers of the congestion control algorithms.

The test network had to fulfill two essential requirements to test our criteria: variable propagation delay and an adjustable drop rate. For this, the standard dumbbell topology was extended as described in the following, and as can be seen in Figure 5. The extended topology features a packet drop inducer and a delay inducer.

To induce delays, we used a dedicated, state-owned research network of the bwNET100G+ Project, which is equipped with configurable 10x10 Gbit/s connections between research institutions in Ulm, Tübingen, and Karlsruhe to achieve variable propagation delay. The connections between the universities were used to form 4 ring structures between the locations. The rings begin and end in Ulm at the patch panel directly connected to the switches used for the tests. There are 38 transceivers but no switches within one 534 km long ring yielding a propagation delay of 6 ms. Connecting all four rings, therefore, enables us to induce up to 24 ms of real physical delay in our tests. The network is isolated, which means there is no traf-

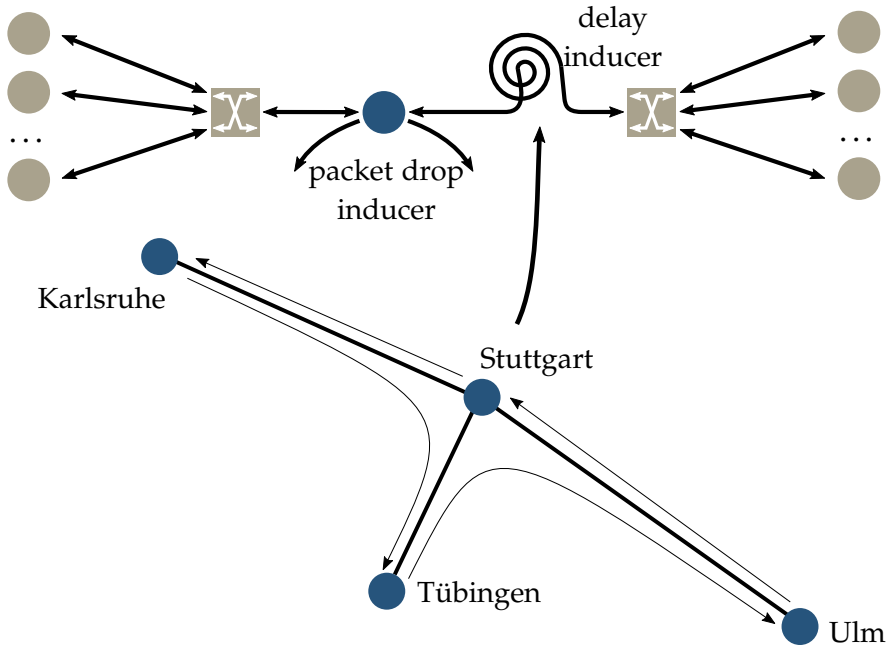


Figure 5: Extended Dumbbell Topology including the Ring configuration in the BelWü research network used as delay inducer. Partially based on [1], © 2016 IEEE.

fic other than the test traffic on the network during the tests. This network setup experiences rare bit flips, which in turn cause retransmissions of TCP packets. Those retransmissions can be observed 15 times on average, with a standard deviation of 2.8 within a 15 min test at 10 Gbit/s. Therefore, the likelihood of a loss to occur is 2.2×10^{-8} for every packet. For comparison: to comply with 802.3ae [152], the error rate of optical fiber connections cannot exceed 1×10^{-12}. In the following sections, we clearly document when this error has an impact on the quality of the results.

A NetFPGA 10G Card as a standalone bump-in-the-wire device was used to realize the adjustable drop rate (packet drop inducer). The reference NIC implementation of the NetFPGA project was used and adjusted to our needs[8]. In its standard configuration, the card is a simple forwarding device at line speed. Our extension makes it possible to set an evenly distributed likelihood for a packet to be dropped.

The NICs are able to send 9.5 Gbit/s without using jumbo frames and are only able to send full 10 Gbit/s with jumbo frames activated. Unfortunately, jumbo frames are not supported by the NetFPGA. Therefore, all tests were conducted with a maximum throughput of 9.5 Gbit/s per NIC.

All in all, as every permutation of variant, RTT, and drop rate or second variant was tested, and every test was conducted five times, a total of 1,280 tests were undertaken, as can be seen in Table 2.

8 Access can be requested at https://github.com/NetFPGA/.

Table 2: Test overview; every listed parameter permutation was tested.

Criterium	Metrics	Parameters				#
Responsiveness	Average throughput	Variant: *Reno, Scalable, HSTCP, BIC, CUBIC, H-TCP*	x	RTT: *0.2 ms, 6.2 ms, 12.2 ms, 24.2 ms*	x Drop rate: *0 or* $10^{-i}, i \in \{7,6,5,4,3,2\}$	840
Efficiency	Throughput distribution					
Fairness	Jain's Fairness Index, Link utilization					120
Downwards compatibility						110
Convergence time	Convergence time, Spread				Variant for 2^{nd} flow: *same as first variant or Reno*	220
						Σ 1,280

4.4 RESULTS

Figure 6 shows the responsiveness of the different TCP congestion control algorithms dependent on round trip time and induced packet drops. The packet drop rates that are used as the basis for the evaluation of the efficiency are highlighted with a gray background. Figure 7 shows the efficiency of these selected drop rates at which the algorithms showed the most diverse behavior concerning responsiveness.

Testing the *responsiveness* and *efficiency* at an RTT of 0.2 ms and without packet drops, every TCP congestion control algorithm fully utilizes the link. Even with drop rates of up to 1×10^{-6}, the average path utilization does not fall under 9.48 Gbit/s. At a drop rate of 1×10^{-5}, however, Reno and CUBIC show significant performance losses. CUBIC shows the same behavior as Reno because it operates in legacy mode and thus behaves just like Reno in this scenario. The other variants perform significantly better with a $cwnd$ that is at least 10 times bigger. At a drop rate of 1×10^{-4} packets, it can clearly be seen that BIC works best in lossy networks with low latency and reaches 8.3 Gbit/s.

Without additional delay in the network, all variants are equally efficient and show low scattering without packet drops. Differences occur only at 1×10^{-5} and above when CUBIC and Reno start to perform worse than the other variants. Q(0.5) is 8.6 Gbit/s, Q(0.25) 8.4 Gbit/s and Q(0.75) is 8.8 and 8.9 Gbit/s respectively. They both show the same behavior as CUBIC operates in legacy mode. At a drop rate of 1×10^{-4}, CUBIC and Reno show significant scattering of 1.5 Gbit/s between 4 and 5.5 Gbit/s; 25% of all values lie above 4.8 Gbit/s.

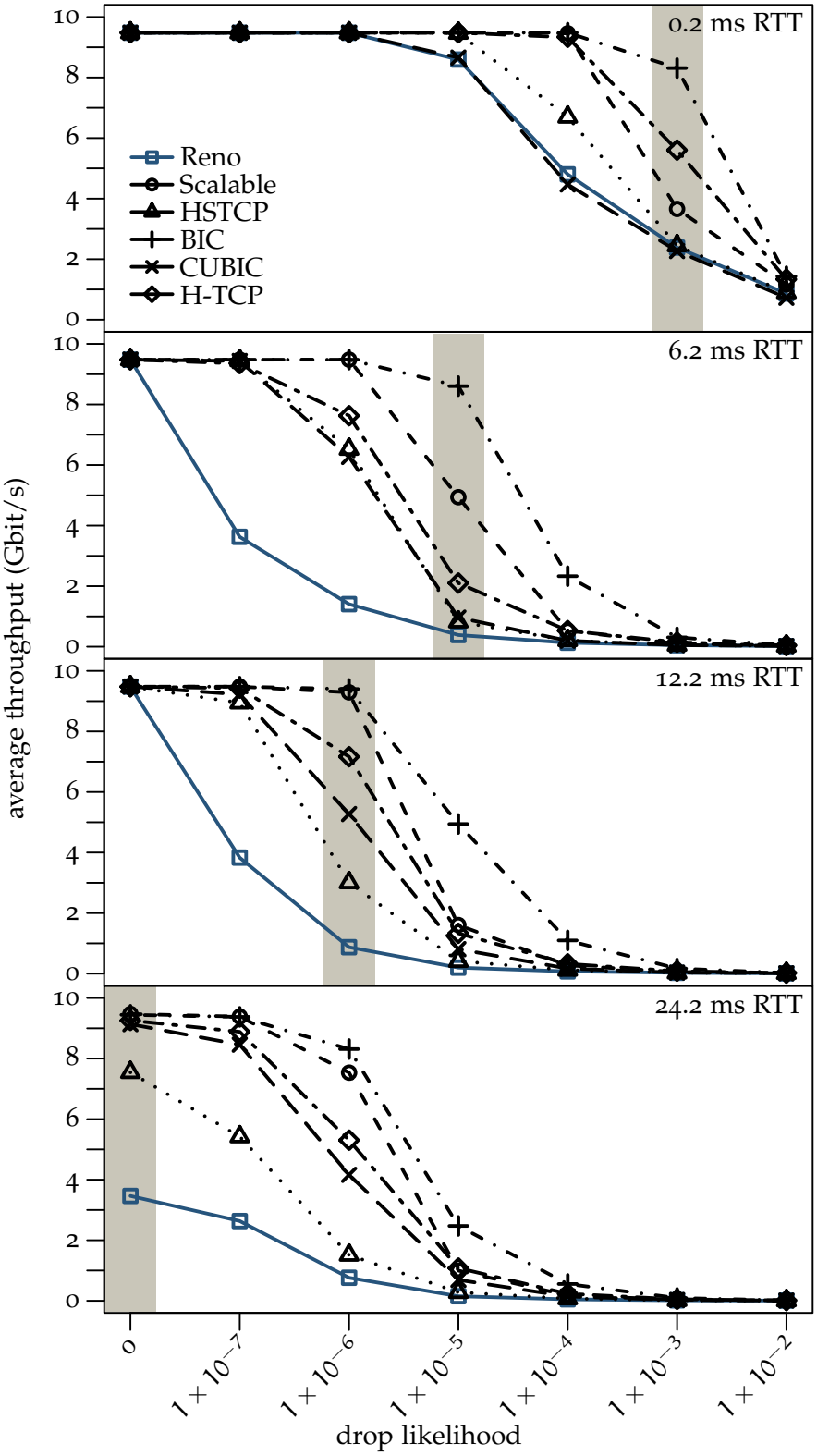


Figure 6: Responsiveness at different induced drop rates. © 2016 IEEE [1].

Concerning efficiency, the differences between congestion control algorithms become apparent at a drop rate of 1×10^{-3}. Reno, CUBIC, and HSTCP show the same behavior with bandwidth scattering between 1.9 and 2.7 Gbit/s. BIC performs best and reaches up to 8.5 Gbit/s with 50% of all measurements between 8.2 and 8.27 Gbit/s.

With an RTT of 6.2 ms, different behavior than with 0.2 ms can be observed with drop rates above 1×10^{-7}.Reno reaches only 3.6 Gbit/s while the high-speed variants reach the uppermost limit, except for of H-TCP with 9.35 Gbit/s. At a drop rate of 1×10^{-6}, Reno reaches 1.4 Gbit/s. CUBIC (6.26 Gbit/s) and HSTCP (6.52 Gbit/s) also show significant performance drops, despite not performing in legacy mode. H-TCP with a similar *cwnd* still shows better performance than the aforementioned variants as the algorithm raises the bandwidth faster after a reduction. Note that BIC outperforms all other algorithms. Even at a drop rate of 1×10^{-2}, BIC still reaches an average throughput of 2.3 Gbit/s while every other variant lies below 1 Gbit/s. BIC's excellent performance can also be observed when looking at its efficiency at 6.2 ms RTT at a 1×10^{-5} drop rate.

At 12.2 ms RTT Reno loses bandwidth even with low drop rates quite drastically. The other variants show very diverse reactions. At 1×10^{-5}, only BIC can keep the bandwidth high with 4.9 Gbit/s on average. The runner-up, Scalable, only reaches 1.6 Gbit/s.

Reno only reaches its rather low throughput because of high throughput at the beginning of the test, because with very low drop probability, it takes several seconds for the first drop to occur. After the first drop, Reno only reaches between 949 Mbit/s and 1.8 Gbit/s at 1×10^{-7} drop rate. At a drop rate of 1×10^{-6}, some variants but especially H-TCP and HSTCP show high scatter. H-TCP reaches an average of at least 6.2 Gbit/s in 75% of all measurements. HSTCP reaches an average of 3 Gbit/s. However, 50% of all measurements lie between 1.7 and 2.9 Gbit/s. The highest bandwidth is 9.42 Gbit/s, the lowest 611 Mbit/s.

At an RTT of 24.2 ms, the aforementioned CRC errors in the networks distort the results. Even with no induced drop rate, the network infrastructure errors lead to low throughput for some of the variants. Relative to the other variants, CUBIC's performance improves significantly with higher RTT, which shows that CUBIC is designed with high RTTs in mind. At 12.2 ms, CUBIC reached 71% of the throughput of H-TCP at a drop rate of 1×10^{-6}; at 24.2 ms, it already reaches 78%.

The effects that can be observed at 12.2 ms are even more significant at 24.2 ms. The network CRC errors also take their toll. Reno loses bandwidth solely because of the scarce CRC errors and the subsequent retransmissions and fluctuates heavily between 742 Mbit/s and 9.48 Gbit/s. HSTCP is also highly influenced by the errors. 50%

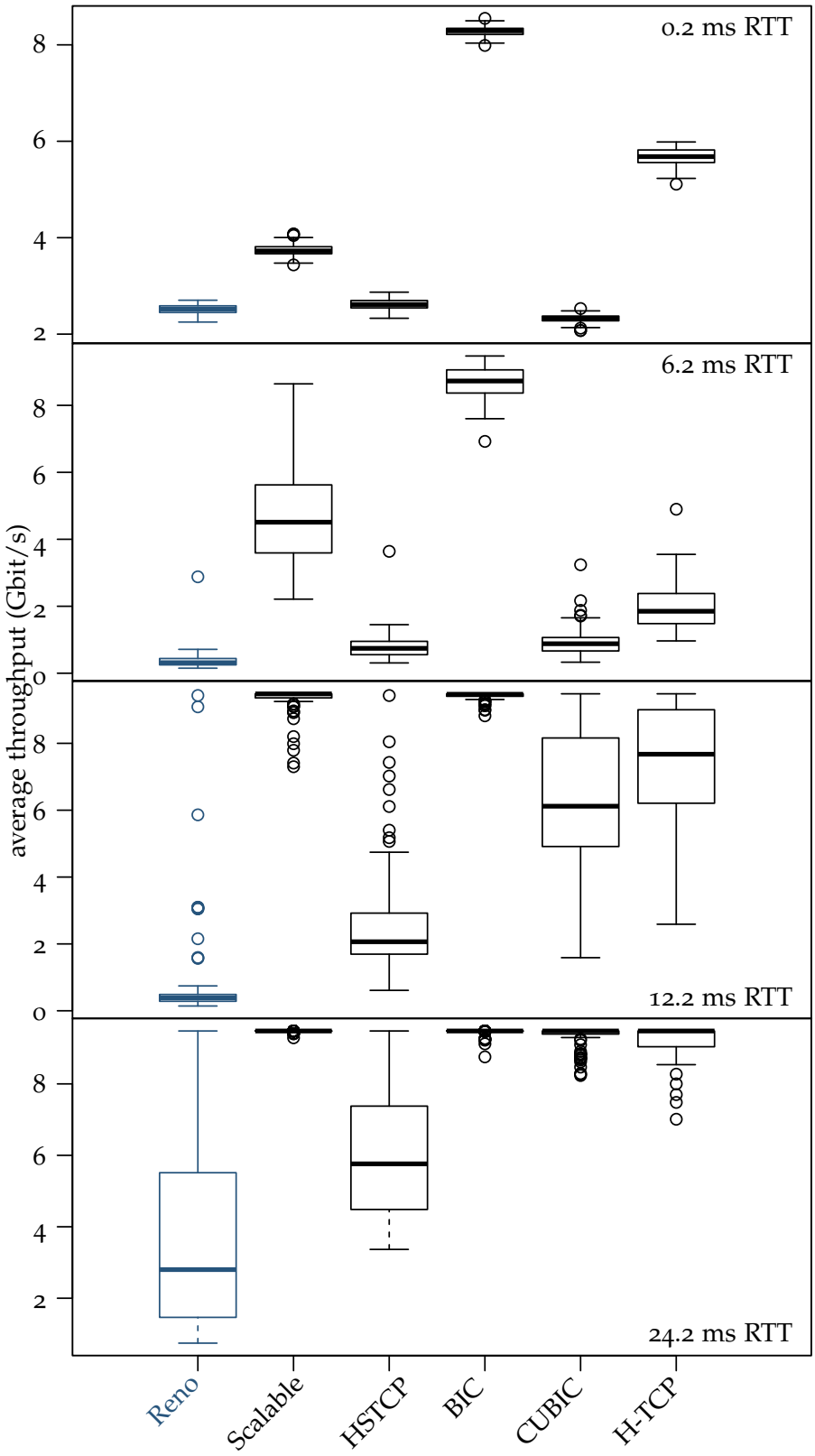


Figure 7: Efficiency at different RTTs and drop rates. © 2016 IEEE [1].

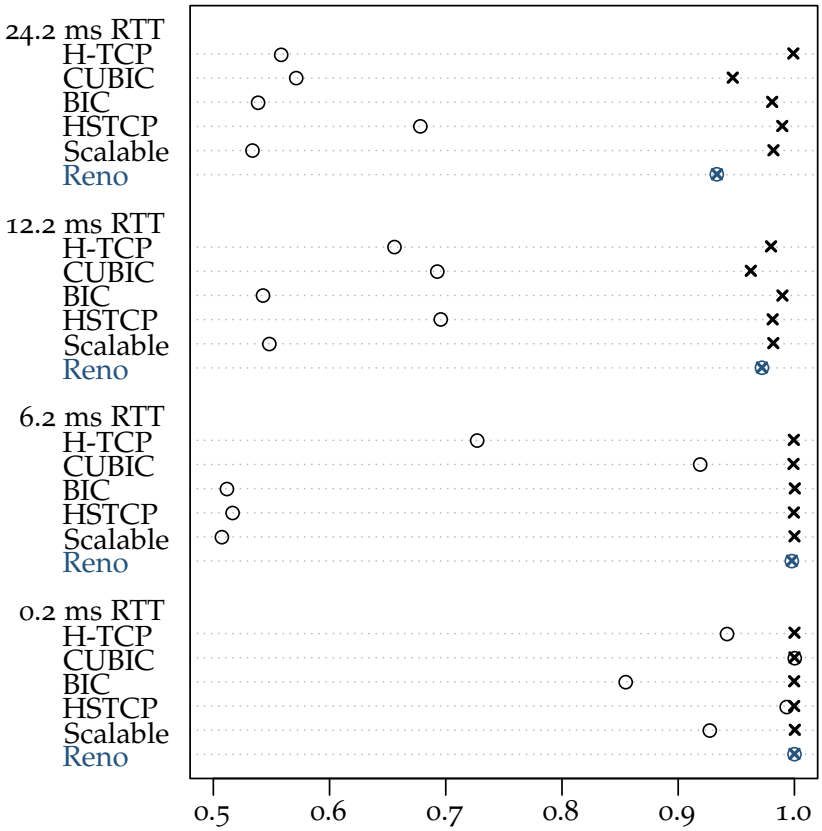


Figure 8: Fairness of the variants against themselves (x) and downwards compatibility against Reno (○). © 2016 IEEE [1].

of the measurements show an average of 4.5 to 7.3 Gbit/s. Outliers can be observed between 3.3 Gbit/s and 9.48 Gbit/s.

It comes with no surprise that in a local network with no additional latency (0.2 ms) — shown in Figure 8 — all TCP congestion control algorithms show complete *fairness*. The biggest difference between the two flows is 150 Mbit/s, and the two flows utilize the ring fully (Figure 9, left). Even legacy Reno performs as well as the other variants. The bandwidth utilization is lowest for CUBIC with 9.29 Gbit/s and best for BIC with 9.42 Gbit/s for both flows combined. At 12.2 ms, fairness lies between 0.96 (CUBIC) and 0.989 (BIC). The very good fairness values come from the flows increasing their bandwidth in parallel as their algorithms behave identically under those circumstances and as there is no overload for most of the test duration. A short overload period in the beginning when both flows try to send at 10 Gbit/s and long recovery times afterward lead to very low link utilization between 3.6 Gbit/s and 5.6 Gbit/s with all variants. Reno performs worst in this regard, but the other variants HSTCP, H-TCP, and CUBIC, show little improvement. BIC and Scalable perform best. With even longer recovery times, the link utilization becomes worse with an RTT of 24.2 ms. Here, H-TCP performs a lot better than before compared to the other variants. Except for HSTCP, all variants

show more than two times the link utilization of Reno. The logarithmic functions of HSTCP seem to work badly with high latency.

The resulting fairness is worse for every permutation when the different variants are not competing with themselves but with Reno to evaluate *downwards compatibility*, which can be seen in Figure 9 on the right. With higher RTT, the flow with the high-bandwidth variant always takes more bandwidth than the Reno flow as the algorithms have a shorter recovery time after packet drops. H-TCP, CUBIC, and HSTCP are the fairest, which negatively reflects in their results concerning link utilization as a fair bandwidth propagation means more bandwidth for Reno and less bandwidth for the high-bandwidth flows with a, therefore, smaller $cwnd$. Hence, in case of overload, the high-bandwidth flow needs more time to recover, and the link utilization goes down. As BIC and Scalable are less fair, the overall link utilization is higher, which becomes especially apparent with 24.2 ms RTT.

In previous work [195], evaluating the *convergence time*, λ was not clearly specified: we set $\lambda = 0.1$, which appeared to be a good trade-off; a smaller λ leads to a smoother convergence but increases the overall convergence time. It is out of scope for this work to provide an extensive analysis of the effect of this metric parameter (just as we do not change the $\varepsilon = 0.8$ given by Li et al. [195]). Instead, we measured the effect of round trip time and congestion control algorithm on ε-convergence time. In addition, we computed a measure for the stability of this convergence, as discussed in Section 2.4.2. The results are shown in Figure 10, where the bar chart represents convergence time, and the scatter plot reflects the spread. For clarity, we left out the measurements with an RTT of 0.2ms: these measurements converged virtually instantly with very little spread. A significant result is that any high-bandwidth TCP variant leads to a very unstable convergence when streams with Reno are involved. This is due to the fact that the distribution of bandwidth is very uneven (see Figure 8), which amplifies the effects of retransmissions and leads to higher spread. As expected, Reno itself shows unstable behavior as the RTT increases. This is related to the fact that Reno's responsiveness is dependent on RTT, which was one of the reasons the new congestion control algorithms were developed in the first place. Finally, we point out that convergence time goes down as the RTT increases, which is partially due to reduced utilization. This reduced utilization is caused by the CRC failures of the link, as discussed in the setup.

4.5 DISCUSSION

It becomes evident that, when there is no packet loss in the network, the actual TCP variant has no substantial influence on the performance. However, as soon as there is packet loss — especially with

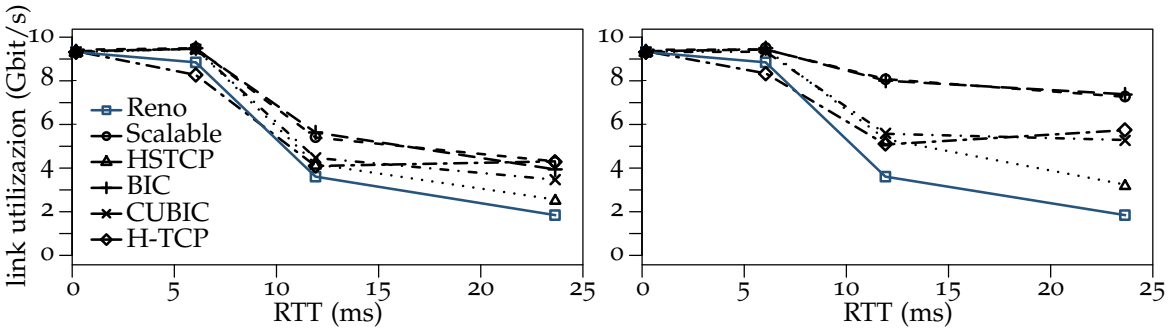


Figure 9: Link utilization of different TCP congestion control algorithms, variant against itself (left) and against Reno (right). © 2016 IEEE [1].

noticeable RTT — the congestion control algorithm affects the quality of service — for example, link utilization and responsiveness — immensely.

Taking into account our use cases mentioned before, BIC shows the best properties for intra-data center and inter-data center communication alike. Especially for higher drop rates, BIC outperforms every other variant. In every test, BIC shows the best behavior or — in case of link utilization and fairness against itself — is one of the best variants.

However, when looking into backward compatibility, BIC shows its weakness. Like HSTCP and Scalable, at an RTT of 6.2 ms, BIC dominates the network link. Here, CUBIC is by far the best alternative, which also performs well in the other tests. At an RTT of 12.2 ms and above, fairness becomes less critical as low link utilization leads to a network without overload and, therefore, sufficient bandwidth for every user.

The decision as to which variant is best for a network cannot be based on a general answer. It depends on the amount of influence one has over the network. For example, if the network is within a data center where every system is under control of the local administrators, our analysis shows that BIC should be used on every system. However, the variants we investigated are all developed with downwards compatibility to Reno in mind, and other variants that were not part of our analysis might perform better in this scenario. On the other hand, CUBIC is the best choice if the composition of the network components is not known, and older systems might still be present, as it offers a reasonable trade-off between link utilization and efficiency on the one hand and also fairness towards other systems on the other hand.

While all modern variants generally outperformed TCP Reno in a 10G setting, the comparison among the other variants yielded more varying results. BIC leads most of our results in the test setup due to its aggressive behavior. At the same time, BIC performed poorly at backward compatibility, and its applicability in higher-latency and

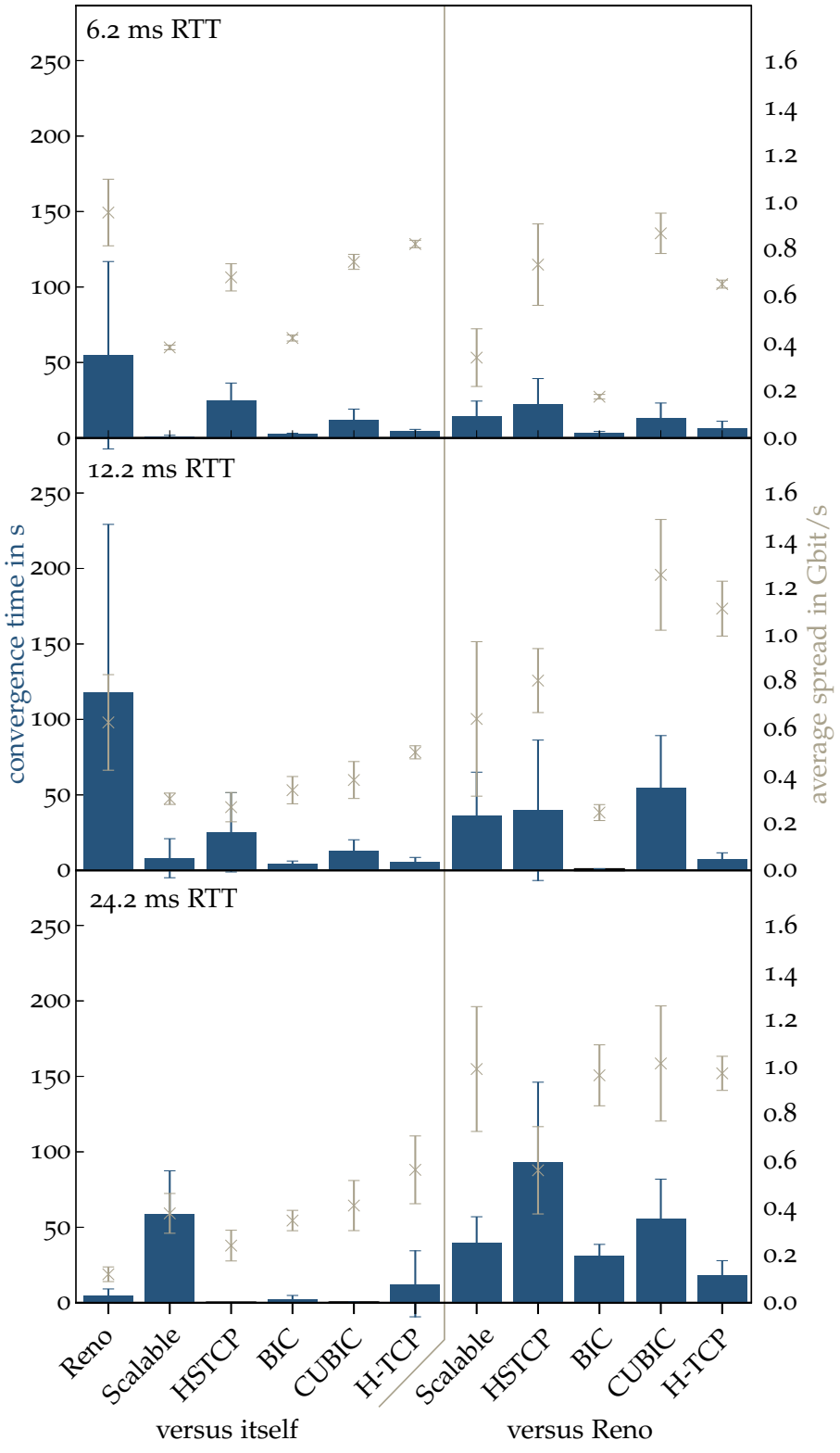


Figure 10: Convergence time and spread of the converged flows. © 2016 IEEE [1].

heterogeneous networks should be considered carefully. For such networks, CUBIC may represent a more appropriate alternative. In summary, we recommend switching to a modern TCP variant for 10G networks and selecting a variant based on the predominant latency and drop rate characteristics that the networked applications will experience in that network.

In general, prior optimizations of the network stack are urgently required to be able to use full bandwidths in the tests. This includes, but is not limited to, network settings of the OS kernel, NIC configurations, and settings of network equipment used in the test topology, for instance, advanced features of Ethernet switches.

If not considered carefully, one single subpar setting of the network setup can have huge effects on the actual performance of the TCP variants in a 10G network. With a change of bandwidth in an order of magnitude also comes a multiplying effect of any uncontrolled parameter confounding the results. ions

4.5.1 *Conclusion*

The evaluation of TCP congestion control algorithms showed a large gap in research concerning network evaluations in high-bandwidth networks. Many things that need to be considered for these evaluations were not documented before, such as the settings necessary to achieve the fairest results in the evaluation. The insights we have gained here helps us to develop the requirements for the necessary evaluations of the other parts of this work. While TCP congestion control tests are comparatively simple (e. g., using a standard the dumbbell topology compared to more complex setups, sending as fast as possible with only one or two flows), more complicated test scenarios necessitate even more deliberations. However, this is necessary when performing more complex evaluations, such as the evaluation of Intrusion Detection System performance. Here, realistic traffic patterns are necessary in addition to malicious traffic generation. For this, a standardized network testing framework that can perform arbitrary tests, including modeling realistic traffic scenarios and behavior models, is required.

5

THE GENERAL PURPOSE NETWORK TESTING FRAMEWORK

An earlier version of parts of this chapter has been published at IEEE LCN 2017 [9].

Leonard Bradatsch has contributed to the design and implemented GPNTF as part of his master project [16] and master thesis [17].

Testing network devices requires realistic network traffic. The devices — especially security devices such as IDS — should be faced with traffic they usually are faced in operations as well as non-typical traffic behavior such as large peaks. Additionally, for future-proofness, network devices need to be tested against probable future network trends such as new protocols, changing user behavior, or new attack scenarios. Devices need to be analyzed in a variety of contexts; using only one or few data sets can be problematic if specific features of the deployment network are not contained in the data sets.

Many of the data sets that can be used for network testing are either old, not guaranteed to be attack-free, specialized for specific network environments, or hard or impossible to gain access to. For anonymization purposes, payload data is missing in most cases. In the anonymization process, additional relevant information might also vanish, such as subnet affiliation or exact timestamps. An additional constraint is that there cannot exist a data set showing future networking trends. For this use case, simulations have to make do.

Simulated data sets are usually limited to the possibilities of programs such as iperf3, that produce traffic flows as fast as possible with no regard to realism. This is acceptable for basic stress testing or performance tests, for example, in our TCP congestion control algorithm evaluation. However, for many more extensive tests, it is doubtful that this can show how a device or protocol would fare in a real network. A system is needed that can produce traffic mixes that can produce realistic protocol traffic patterns. Our goal is to achieve a traffic mix as close to Internet traffic as possible with reasonable defaults based on prior analyzes of Internet traffic while offering high customizability. This allows for analyses on how changing behavior in the Internet — for example, based on future trends such as an increasing amount of video streaming, new attacks, or new protocols — influences performance of the systems under test.

Therefore, we introduce the *General Purpose Network Testing Framework (GPNTF)* meant to fill a gap in network testing environments and current possibilities. GPNTF is able to produce traffic based on traffic models of real network traffic. It can simulate both benign and malicious traffic in a configurable mixture of traffic classes of both kinds. A tester can analyze how their network devices and services handle, for example, rising demand for buffered video entertainment

or a rise of Denial-of-Service attacks. The Framework is easily extendable to support future networking technologies.

In the following, we describe how we modeled benign and malicious traffic, how GPNTF is implemented, evaluated the systems, and produced several data sets both with and without GPNTF to provide data for our analyses documented in the following chapters.

5.1 PRODUCING BENIGN TRAFFIC

To implement benign traffic with the biggest coverage of all traffic classes, we are focusing on storage (e. g., Google Cloud, iCloud, and Dropbox) and marketplace traffic (e. g., Windows Updates, Google Marketplace, iTunes), file sharing (e. g., BitTorrent), buffered video entertainment (e. g., NetFlix, Youtube, Amazon Prime Video), and web browsing. According to the Sandvine[1] reports [267–269] these traffic classes make up over 80% of worldwide Internet traffic.

5.1.1 *Web Browsing*

The web is the part of the Internet most users are somewhat familiar with, and for many, it is synonymous with the Internet. It takes up between 10% and 20% of all traffic on the Internet depending on region [267–269]. This is especially noteworthy as web traffic requires a lot less bandwidth per user than the other traffic classes as a big part of the web is text-based. Although HTTP/2 gains significant traction in the last couple of months, HTTP/1.1 is still widely adopted.

Pries et al. [252] and Choi and Limb [98] analyzed and documented how web traffic looks like. The typical procedure for web requests begins with the establishment of a TCP connection between client and server. Within this TCP connection, the clients send an HTTP request, requesting a web page. The main object — typically an HTML document — is sent back by the server. Within this main object, several other resources are linked, the embedded inline objects (e. g., JavaScript libraries, images, cascading style sheets (CSS)). These inline objects do not have to be on the same server as the main object. Often JavaScript libraries or fonts are embedded from services such as Google Fonts[2] or advertisements are embedded from Google Ads[3]. According to Butkiewicz et al., on average, when requesting a web page, data is downloaded from ten different servers [78] (identified by distinct (sub) domains measure on *"roughly 1700 websites from four geographically distributed locations over a 7 week period"*). However, there

1 Sandvine Incorporated produces network equipment and is operating worldwide. They publish reports on worldwide network composition regularly divided by world regions. https://sandvine.com

2 https://fonts.google.com

3 https://ads.google.com

is no information on how the objects are distributed over the different servers. It is also possible that the website operator distributes their resources over several of their own machines. Follow and like buttons hosted on third-party servers are often implemented as additional main objects. The average website consists of 2.19 main objects. However, the mode (i. e., the most common value) is still clearly one main object, as 60% of all web sites have only one main object [252] (in the Alexa top one million websites by number of visitors). Usually, browsers open several TCP connections to one server. While the HTTP/1.1 standard recommends up to two TCP connections to the same server, most browsers open many more[4]. As HTTP/2.0 features multiplexing within one TCP connection, this is no longer necessary, and only one TCP connection needs to be opened. Additionally, the parsing time denotes the time between the arrival of the main object and the arrival of the first inline object. HTTP supports compression (in both versions currently used). It is in use by roughly 80% of all websites mostly done with gzip according to one study based on the Alexa top 10 million by number of visitors [315]. However, there is no information available to what extend these websites use compression. Between requests, users tend to read the website before requesting a new one. This idle phase has to be taken into account when modeling user behavior. Moreover, browsers tend to cache inline objects. For example, CSS files of commonly visited websites are usually only loaded once and then kept.

In our model, we need to take the following parameters into account:

- request size
- main object size
- amount of main objects
- parsing time
- number of inline objects
- inline object size
- number of TCP connections
- user reading time

The parameters and their values are taken from the aforementioned resources [78, 98, 252, 315], and the full list can be seen in Table 16 in the Appendix listing the literature values and how they are implemented in GPNTF. As there is no sufficient information available, the following parameters could not be considered for the model:

4 Firefox opens up to six by default, which can be checked and changed in about:config. Parameter name: network.http.max-persistent-connections-per-server

- number of servers (set to one)
- multiplexing on HTTP/2.0
- HTTP compression
- client-side website caching

5.1.2 *File Sharing*

While file sharing's share of the Internet traffic is declining in many regions of the world (especially in North America following the emergence of streaming platforms), 30% of Asia's Internet traffic still consists of this traffic model [267].

File sharing is a peer-to-peer approach to distribute data or information between networking nodes. There is no central server in this model — in contrast to the other traffic classes. Data is shared between participants often without central management instance or minimal central management, and all peers might download or upload data to other peers. As users in peer-to-peer networks are usually private users, a user downloading data from a peer-to-peer file sharing service usually downloads from several peers simultaneously to spread the load over many participants to not overload the limited upload bandwidth of a single participant. For this, the data is partitioned in chunks, and these chunks can then be obtained from different peers. For this, an interest message is sent to the other participants for specific chunks of a file. The hit rate indicates how often this is successful (i. e., it denotes the chance that a specific chunk is available at a specific peer). Simultaneously, each participant listens for requests of the data they offer to distribute.

The central parameters for this traffic model were analyzed by Basher et al. [58]. They mention flow size, flow inter-arrival time and flow duration, geographic distributions of the peers, and the number of concurrent flows as *"characterization metrics"* for file sharing services. In addition, Basher et al. also reported on the amount of concurrent upload and download connections and the amount of connections to peers with distinct IPs. Most other analyses focus solely on BitTorrent, as it is by far the most common peer-to-peer file sharing system [276]. Le and But [185], and Calchand et al. [82] analyzed packet size and Le and But also the ratio of packet sizes. Additional to these sources, file sizes are of interest. Afridi et al. [42] analyzed the file sizes of video files in BitTorrent network. These values are also used by us as a source for file sizes in peer-to-peer networks. All in all, the following parameters are considered in our model:

- file size
- flow size

- packet size
- amount of concurrent flows
- amount of connections to peers with distinct IPs
- number of upload connections

The following parameters mentioned in literature were omitted in our model implementation as they are network dependent or no reliable information could be found:

- chunk size distribution
- flow inter-arrival time
- flow duration
- geographical peer distribution
- bandwidth limitations
- hit rate
- idle time duration i. e., the duration of the time between a client participating
- share of *freeriders* i. e., users that only download but do not upload
- tit-for-tat policies i. e., are freeriders sanctioned in any way

The full list of parameters can be seen in Table 17 in the Appendix.

5.1.3 *Buffered Video Entertainment*

Similar to web traffic, video streaming sites often use the HTTP protocol to transfer their data to the client. However, the traffic pattern differs greatly [62]. Video streaming basically refers to the process of downloading a video to a client while simultaneously playing the video. As throughput varies by client connection quality and might also change over time, the video stream is buffered at the client to compensate for short download interruptions or short slumps in throughput. Additionally, as many different quality levels are possible both for video (e. g., full HD, 4K, 480p, etc.) and audio (e. g., 128 kbps, 160 kbps, or 192 kbps), the quality can be dynamically changed depending on the capabilities of the client and the throughput of the network. This method is called *adaptive video streaming* and is deployed by most popular video streaming platforms. The service holds videos in short segments of different quality levels and a certain segment length. At the beginning of the stream, the client receives a manifest file describing the meta-information about the video

in question. Clients can request a new segment choosing the quality level from the manifest file that its current connection allows (with a certain margin to assure reliability). A widespread video streaming standard is MPEG-DASH. It uses the XML-based MPD standard for the manifest file.

Laterman et al. [183] and Reed and Aikat [256] analyzed popular video-on-demand and live streaming platforms (Netflix and Twitch) and reported the used protocols, connection durations, volume and number of connections per stream, data rates, segment request rate, segment downloading approach, and length of connections. Biernacki [62] used NetEm, the Apache web server, and VLC[5] to analyze average throughputs and bandwidth oscillations when the video stream is faced with packet loss in the network.

The observed parameters implemented in GPNTF are:

- video length
- HTTP version
- segment length and size
- manifest
- segment downloading approach
- segment requesting rate
- adaptive streaming algorithm

The full list of which parameters were chosen can be seen in Table 18 in the Appendix. Video quality is dependent on the available bandwidth. VLC supports the MPEG-DASH standard fully and is therefore used to produce the video streaming traffic by embedding it in GPNTF. One downside of using VLC is that it does not support digital rights management, which consequently is not part of the GPNTF implementation.

5.1.4 *Storage & Marketplace*

Storage and marketplace traffic are two traffic classes that are very similar to each other. Storage describes cloud storage providers such as Dropbox[6] and the data syncing between the provider and the client. Another non-proprietary service of this class would be FTP. Marketplace describes content providers such as Apple iTunes or the Google Play Store, where for instance, games, apps, videos, or music are available to users. The biggest difference between the services is that

5 https://videolan.org
6 https://dropbox.com

marketplace usually does not feature any uploads except for requests sent to the service providers.

Kim et al. [173] analyzed traffic patterns of this traffic model. The services differ by average file size, usage of application-layer protocol, number of connections, and packet sizes. The packet sizes are usually fixed during a session. Traffic is unidirectional during the data transfer. Here, reliable data can be found for the FTP protocol [173] and the parameters in GPNTF are set based on this analysis:

- packet size
- file size
- data and control connections
- idle phase duration

Unfortunately, no data can be found for marketplaces. Therefore, GPNTF uses the same parameters as for storage. All parameters are variable; any GPNTF user can set different values as soon as reliable data can be found. The implementation in GPNTF is done based on iperf3. The full list of parameter values can be found in Table 19 in the Appendix.

5.2 PRODUCING MALICIOUS TRAFFIC

Philipp Spiegelt [33] has contributed to this section with his bachelor thesis.

Due to our focus on IDS and DDoS mitigation, malicious traffic in GPNTF is constrained to network-based attacks. The most important attacks for us are reconnaissance, intrusions, and Denial-of-Service attacks. As many open-source tools for these attacks already exist, we choose to integrate them into GPNTF instead of building new programs. These tools are widely used by attackers already and, therefore, already represent realistic attack traffic.

5.2.1 *Reconnaissance*

Reconnaissance is not an attack per se; however, it is usually a precondition or at least a sign for a looming attack. Therefore, network administrators are well-advised to keep an eye open for reconnaissance activity in their network. The most common form of reconnaissance are port scans. We use the classification of port scans by Lee et al. [187] described in the following.

The simplest form of scan is the PING scan based on ICMP. Simply sending ICMP echo-request packets to the target usually triggers a response from the target (ICMP echo-reply). As this method is also used to deduce if a target is up and reachable by network administrators and other power users, singular instances of ICMP echo-request

packets are no sign of an attack. However, a high number of these packets can be an indicator.

TCP scans are a group of network scans based on the TCP protocol. The SYN scan tests if a port is open. If a *SYN* packet is answered by a SYN/ACK, an open port is detected that accepts connections. An RST packet is interpreted as a closed port. During a *connect scan*, the connection is fully established, which makes it easier to detect. The *FIN* scan uses undocumented TCP behavior inherent in most systems. When a FIN packet is sent to an open port, no reaction from the server is expected while on a closed port, an RST packet is sent back. *Null* or *Xmas* scans are variants of the FIN scan, where none or all the TCP flags are set (*"Lighting up the packet like a Christmas tree"*). Firewalls that filter for SYN or ACK flags can be bypassed.

During a *UDP* scan, the scanner sends UDP packets to a target port. On closed ports, the target should respond with an ICMP port unreachable packet; other ICMP packets indicate some kind of filter in front of the port. No response can mean that packet is blocked, lost, or in most cases, that the port is open.

Remote operating system (OS) detection usually works with fingerprinting the behavior of the target. Basically, five different methods can be used [310]. For one, *banner grabbing* can identify an OS by analyzing service banners typical for the OS run by the target. *TCP segmentation* and *ICMP response analysis* exploits that some behavior for these protocols is not standardized, and operating systems behave differently. The same is true for the *Initial Sequence Number (ISN)* analysis of TCP. Furthermore, there are some *specific Denial-of-Service attacks* that only work on specific operating systems and can be used to see if the target is susceptible to those.

Nmap[7] can be used to implement all of the port scanning techniques introduced here and is therefore integrated with GPNTF.

5.2.2 *Intrusions*

Intrusions can be achieved with different techniques.

One of them is *exploitation*, where unintended behavior of a system is exploited to gain access to this system. A zero-day exploit is an exploit unknown to the developers (the developers have known about this attack for zero days). As zero-day exploits are by definition not publicly known, a traffic generator cannot generate traffic simulating zero-day exploits. However, many former zero-day exploits can be simulated. One example of this is the *EternalBlue* exploit that exploits three bugs in the Windows implementation of the Server Message Block protocol (SMB) for file sharing. It became widely known outside of the security community through the *"WannaCry"* ransomware attack. As this attack is one of the most common network attacks cur-

7 https://nmap.org

rently, we chose to implement it in GPNTF. For this, we use Metasploit, the Metasploit API, and the Metaspliotable VMs[8] (Ubuntu 14.04 and Windows Server 2008).

The *brute-force* login attack tries to gain access to a system by trial and error approach. A database with common passwords is usually used, combined with a database of known or guessed user names. The attack can be executed with Metasploit (*"The world's most used penetration testing framework"*, according to its creators[9]), for example for SSH or FTP and can, therefore, be used within the GPNTF.

Another method is the *Man-in-the-Middle Attack*. The adversary eavesdrops on the communication between two targets. They relay messages between the targets, impersonating the other communication partner, respectively.

Phishing attacks are mostly done by e-mail where the attacker tries to get the target to provide access to a system or reveal information.

The Man-in-the-Middle and Phishing attacks are not currently implemented in GPNTF as they are not necessary for our purposes yet. However, the extension is planned for the future.

5.2.3 *Denial-of-Service*

In terms of Denial-of-Service attacks, an extensive analysis can be found in Chapter 9 describing the attacks in more detail. There is a large variety of attacks. We focus on the most common, current attacks. In terms of *flooding attacks* and *reflective attacks*, GPNTF uses hping3. Slow DDoS attacks are also implemented with the tool SlowHTTPTest, as it already supports both the slow GET and slow POST attack.

5.3 IMPLEMENTATION

The GPNTF design, as shown in Figure 11, is able to run in three modes — as a network client (1), a network server (2), and as a controller for administration purposes (3). Only the client and the server mode are part of the current implementation. All modes can be configured by configuration file that can later also be used as documentation of the chosen evaluation scenario.

Running in client mode, GPNTF is able to simulate diverse real network clients with different fields of application. The client mode is built upon three main modules: the client task handler module, the L4 modules, and the L7 modules. The client task handler module reads all necessary client parameters from a configuration file for generating the requested flows and coordinates the construction of L4 and L7 objects for layer 4 and layer 7 protocol headers. In case

8 https://github.com/rapid7/metasploitable3

9 https://metasploit.com/

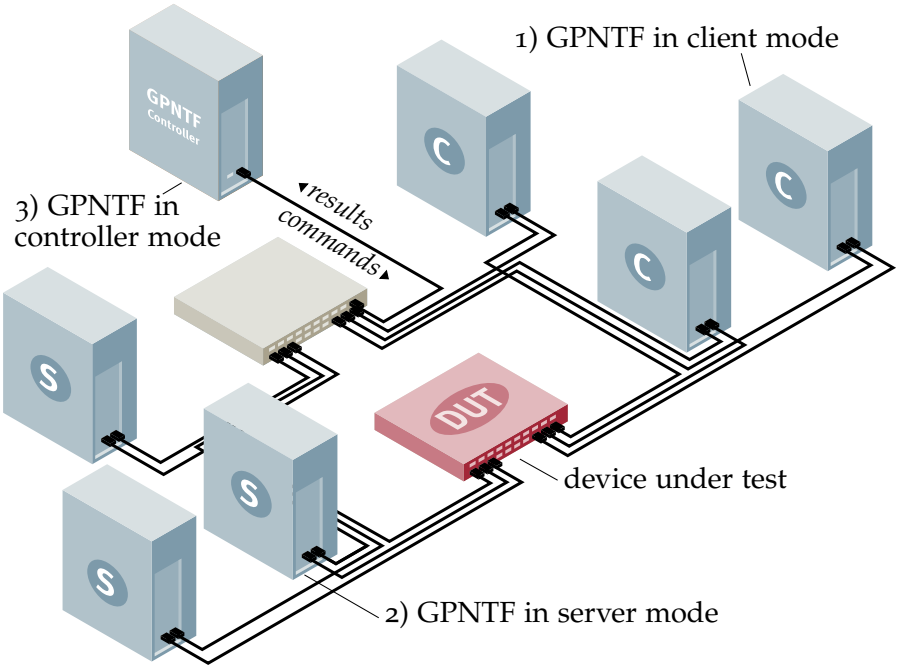


Figure 11: GPNTF architecture with three clients and three servers testing one physical device.

of an external solution (such as VLC, Metasploit, and iperf3), the client task handler executes these programs accordingly. Moreover, the client listens at the adjusted ports to be able to handle incoming server responses. Incoming packets are checked against the states of an internal state machine. Based on the result, the packet builder can build the correct answer. The client state machine is implemented in a client state machine module to which it is possible to connect new state machines handling new protocols.

Running in server mode, GPNTF simulates the behavior of a real network server. The server mode contains the same modular design as the client. The server task handler module opens the user requested server sockets and processes incoming client requests. Just as in client mode, GPNTF must check the incoming packets against the states of a server state machine module and must determine the correct response based on the specific client request.

Our proposed architecture provides an option to control all connected clients or server instances remotely. For this purpose, the controller sends a configuration file to each declared client or server instance. In return, the targeted instance responds with the captured pcap file and the stored statistical information. As the central controller instance was not necessary for our tests, we decided to postpone its implementation and focus on the client and server modes' implementation.

There is no upper limit for the number of network nodes or the network devices within the network. GPNTF, in principle, runs on any

node with a modern Linux operating system (tested on Ubuntu), sufficiently strong hardware to accomplish sending rates necessary for the testing network, and a NIC. The required performance of hardware and NIC depends on the specific test cases.

GPNTF is published under the GPLv3 license and available at Github[10].

5.4 EVALUATION

Philipp Spiegelt [33] has contributed to this section with his bachelor thesis.

There are two aspects we want to investigate in this evaluation. First, correctness of the implementation and second giving an idea of how long a test has to be to give representative results. As many of the benign traffic generation parameters depend on distributions, a minimum set of runs or a sufficiently long session is necessary to build a representative set of the traffic where the distributions come close to the model. To measure if the implementation fits the model, we use Pearson's Correlation Coefficient.

5.4.1 *Pearson's Correlation Coefficient*

Pearson's Correlation Coefficient is a measurement of linear correlation between two variables. The coefficient can take values between -1 and 1, where 0 means no correlation between the two variables, 1 means total linear correlation, and -1 means total negative linear correlation (Cauchy–Schwarz inequality). For samples of the variables X and Y, the coefficient r_{xy} is defined as follows.

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{7}$$

With sample size n, x_i and y_i individual sample points and $\bar{x}$ (and $\bar{y}$) as the mean of all $x_i \in X$ (and $y_i \in Y$). For our approach, as we compare a sample with a formula, we take the value of the formula at the same position as the sample for the comparison.

5.4.2 *Results*

The length of a test can be set with the session length parameter. For web browsing, it defines how many websites are downloaded. Figure 12 shows how session length influences how close the data output comes to the input model. In this case, the main object size for web traffic is plotted against their probability of occurrence for different session lengths and the Weibull model distribution used as input (as cumulative distribution function, CDF). We can clearly see

10 https://github.com/vs-uulm/General-Purpose-Network-Testing-Framework

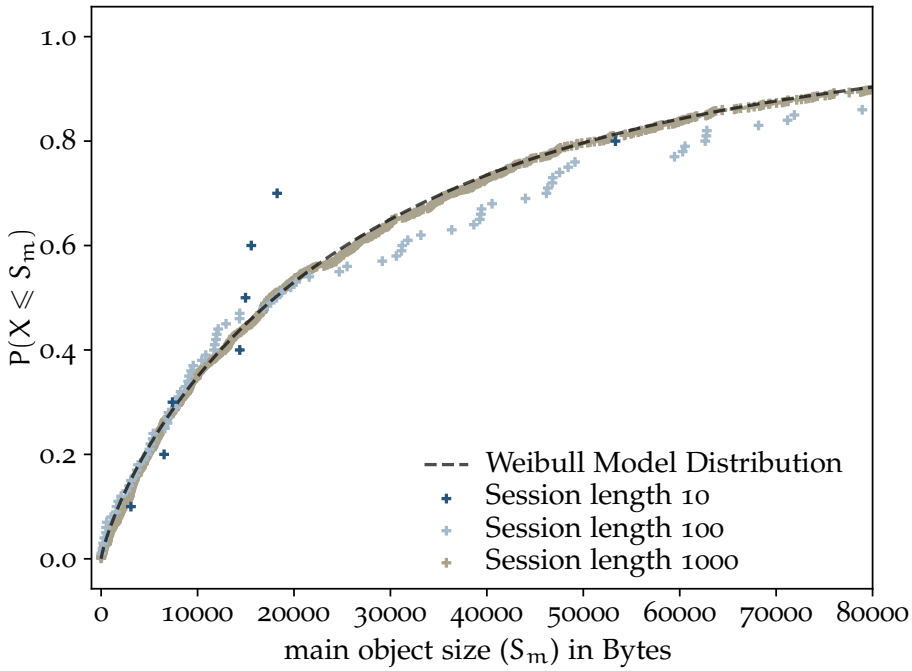


Figure 12: Main object sizes (at different session lengths) plotted as CDF compared to the model distribution. One test run. Based on [33].

how higher session lengths lead to results much closer to the model distribution.

This can also be seen when we take a look at Pearson's correlation coefficient for the different web browsing parameters, as can be seen in Figure 13. Session lengths of 1 000, and 10 000 show very high correlation coefficients for inline object size, main object size, and number of inline objects. This shows for one that the system works correctly and that session length of 1 000 and above guarantee that the model is correctly transferred to the test traffic.

For storage and marketplace traffic, the number of parameters is far more limited. Figure 14 shows the file size distribution compared between input and output measured in the network. For file sizes of 1.5 GiB and below, the measured results are very close to the input. For a file size of 10 GiB, we measure a deviation of 4.6% between measurement and parameter.

For file sharing, we analyzed the flow sizes. Figure 15 shows that while the average correlation coefficient is already high at a session length of 10, it stays the same independent of session length. While most measurements show good results close to the model, some measurements deviate immensely. Therefore, this traffic mode as is can be used to conduct further analyses. However, one should always measure if the output actually matches the model for every measurement.

Both the buffered video streaming traffic as well as the attack traffic models were produced with the use of third party programs. These

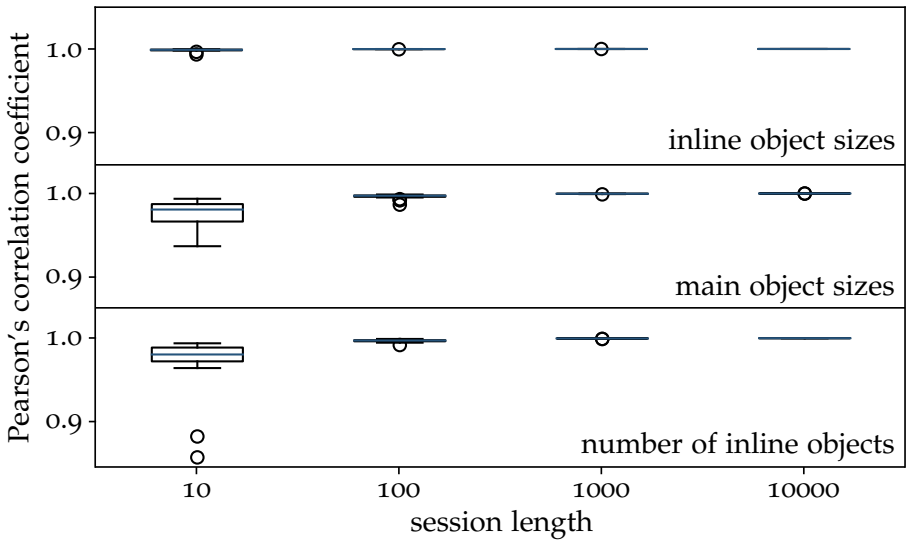


Figure 13: Pearson's correlation coefficient for several web browsing parameters dependent on session size (20 test runs). Based on [33].

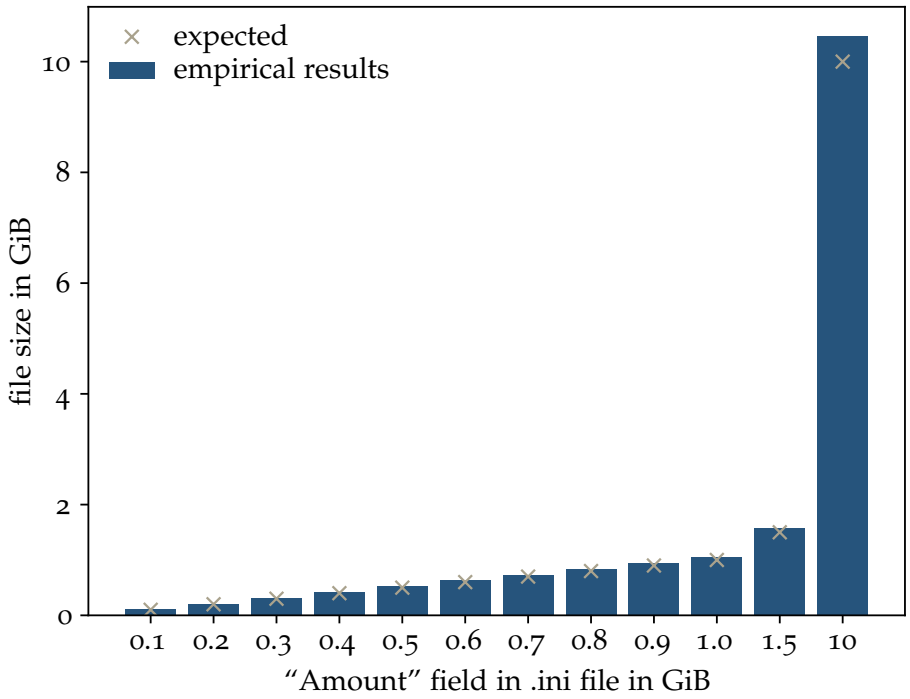


Figure 14: File sizes measured versus data input. Based on [33].

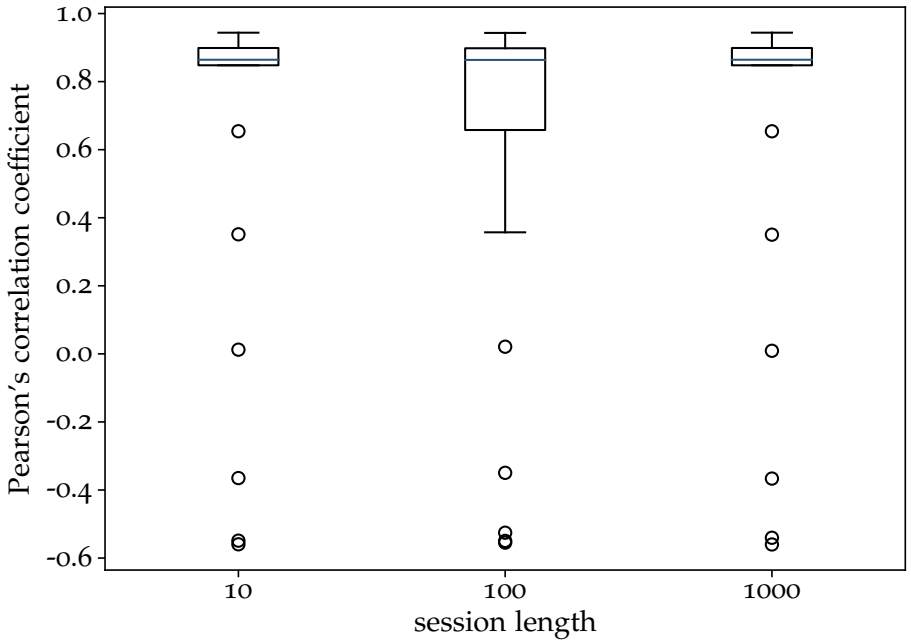


Figure 15: Pearson's correlation coefficient for file sharing flow sizes dependent on session length. Based on [33].

programs were already tested by their original vendors. No extensive evaluation beyond basic functionality tests (that confirmed that the implementations are working) was conducted and was therefore omitted in this chapter.

5.5 PRODUCED DATA SETS

For our evaluations in the upcoming parts III and IV, six data sets were produced. One for each of the four benign traffic classes, one data set to test slow DDoS attacker identification mechanisms, and — for IDS testing — one malicious traffic data set based on the community ruleset of the Snort IDS.

5.5.1 *Benign Data Sets Produced by GPNTF*

The web browsing data set has a file size of 6.9 GiB. It contains 1.64 million packets. 100 clients were simulated, sending requests to 100 web servers downloading the websites.

The buffered video streaming data set takes up 43 GiB in 1.74 million packets. The video file used here is the Blender Foundation production *"Tears of Steel"*. The adaptive video streaming made use of the 101 bit/s bitrate (at the start for each video transmission) and the highest possible bitrate of 10 Mbit/s (otherwise).

The second-largest data set is storage and marketplace, taking up 31 GiB. In 100 flows, 50.7 million packets were sent.

Table 3: Composition of the SUEE data sets benign traffic.

data set	start date (duration)	hosts (external/internal)	internal hosts wifi (eduroam/welcome)
SUEE1	2017-11-02, 24 h	1 634 (1192/442)	243 (97/146)
SUEE8	2017-11-05, 8 d	8 286 (6755/1531)	705 (328/377)

The file sharing data set takes up 12 GiB in 8 215 flows, and 892 410 packets.

All data sets other than the buffered video streaming set contain randomly generated payload data. All other parameters were left at the default values that can be found in the Tables 16 to 19 in the Appendix, the data sets are available for download. Download locations, and more information is available at Github[11].

5.5.2 *SUEE Data Set for Slow DDoS Attacker Identification Testing*

Parts of this section have been published at EAI SecureComm 2018 [5].

Additional to the data sets that emerged from the GPNTF, we also built a data set specialized for the evaluation of slow DDoS attacks. We collaborated with the student union for electrical engineering at Ulm University[12] to record requests made to their web server, one data set containing 24 hours (2nd to 3rd November 2017 with 1,634 clients, SUEE1) and another data set containing eight days (5th to 13th November 2017 with 8,286 clients, SUEE8) of traffic data. The web server of the student union provides information about the union on its main site, public real-time transport information for bus stops in the city which is used primarily on mobile devices via mobile networks, as well as several external and internal services such as a printing service for the lecture notes of the electrical engineering courses and exams.

Both pcap files contain only header data since the data sets were anonymized and do not contain application-layer payload due to privacy concerns. There have been no attacks reported during the times of recording of the benign data sets. The data sets serve the following purposes: SUEE1 is used as training data set. SUEE8 then can be used to determine whether the trained mitigation system is capable of mitigating attacks adequately.

To facilitate this, we have mixed the SUEE data sets with attack recordings from the three attack tools slowloris, slowHTTPTest, and

11 https://github.com/vs-uulm/2019-Network-Data-Sets
12 https://fs-et.de

Table 4: Port distribution of the SUEE data sets benign traffic.

data set	number of packets	TCP source port 80/443	TCP destination port 80/443
SUEE1	2 089 436	747 912 / 173 978	967 623 / 199 923
SUEE8	19 301 217	7 175 627 / 1 229 516	9 312 537 / 1 583 543

slowloris-ng by merging the capture files. We have published the data sets with a more detailed description on github[13]. The MAC and IP addresses are anonymized, i. e., addresses are overwritten. Benign clients' IP addresses in the anonymized data sets are moved to the 192.168.0.0/16 block while attacking clients are in the 128.10.0.0/16 block. The IP addresses count up chronologically after their first occurrence within the data set. The original IP addresses were in part from the Ulm University network and mostly from diverse networks in Ulm and surrounding areas. Table 3 contains the composition of the benign traffic in the SUEE data sets. The same IP address in SUEE1 and SUEE8 are not affiliated. However, every packet sent (or received) by an IP within one data set was originally sent (or received) from the same IP address.

The attacking tools were adapted to allow IP spoofing to simulate distributed attacks and were left in standard configuration apart from that. The parameters for slowHTTPTest were 30 seconds intervals, 8 192 bytes for the Content-Length header, 10 bytes POST-body length per packet, and one socket per client. Slowloris is also configured to use only one socket per client. The default configuration was left in place in all other settings, resulting in a packet interval of 15 seconds.

Slowloris-ng includes several changes to the original slowloris. The additional features implement randomized behavior, which is configured to send in intervals of 15 seconds with a randomization interval of 5 seconds and to send the header lines as bursts of single messages per character.

5.5.3 *Producing Traffic From IDS Rulesets*

Mathias Wagner [39] has contributed to this section with his master project.

Similar to Mucus [229] and GENESIDS [117], we also produced a data set called *the trigger set* by using the Snort community ruleset[14]. Snort rules contain rules for fast string matching and regular expression matching for slower but more in-depth analysis after the strings match. We produced payload data based on the regular expressions and string matching rules within this set. For the string matching, the strings that need to be matched are simply copied into the payload.

13 https://github.com/vs-uulm/2017-SUEE-data-set
14 https://www.snort.org/downloads/community/snort3-community-rules.tar.gz

For the regular expressions, deterministic finite automata (DFAs) are built (following the DFA construction that will be described in Chapter 8). Then, strings matching these DFAs are randomly produced by following random paths in the automata until a finite state is reached. These resulting strings are guaranteed to trigger the regular expression. Some other information such as protocol (TCP, UDP, ICMP, or IP) is also read from the rules; depending on this protocol, a port number might also be necessary, which too can be configured in the rules or is randomly chosen. If an IP address is configured, it is used. Otherwise, a random, valid address is chosen.

With this data, the payload is generated that should trigger the rules. The payload consists of the concatenation of the string matching keywords and the regular expression triggering string. Resulting empty packets (that would still trigger rules because of their metadata) were subsequently removed. If the resulting payload does not exceed 500 bytes, random padding bytes are added.

Packets are then, in turn, constructed with this payload and aforementioned information, sent through a dummy interface, and recorded at another dummy interface. In total, 3 482 rules are triggered by this data set. The trigger set has a size of 190 MiB, 303 800 flows in 349 690 packets as nearly all flows only require one packet to trigger a rule. The data set is available for download. Download location and more information is available at Github[15].

5.6 SUMMARY

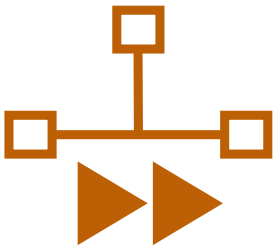
The General Purpose Network Testing Framework is a system that can be used both for live testing and to produce data sets that realistically represent the features of real network traffic. The system can be distributed on several machines and can be run in client and server mode. In this setup, the system can be used for live testing of devices and algorithms in the network. Different classes of network traffic — both benign and malicious classes — can be produced. Models for benign web traffic, file sharing, adaptive video streaming, and storage and marketplace traffic were produced based on literature and our own measurements. Example data sets for these four traffic classes were produced and published. The General Purpose Network Testing Framework is open-source software and publicly available.

In addition, traffic on a web server running at Ulm University was recorded and anonymized with the help of the Student Union Electrical Engineering at Ulm University. Attack traffic from the slow DDoS attack tools slowloris, slowHTTPTest, and an adapted version of slowloris (slowloris-ng) were added to this data set to produce a labeled, specialized data set for slow DDoS testing. Based on the Snort community ruleset, a specialized IDS testing data set was produced

15 https://github.com/vs-uulm/2019-Network-Data-Sets

meant for performance tests of intrusion detection systems. These data sets were published as well.

This system and these data sets made the evaluations of the systems in the following parts of this work possible using the trigger data set and the benign traffic data sets as background traffic for our evaluation of our IDS acceleration implementation in Part III and the SUEE data sets as foundation for our tests of the slow DDoS identification mechanisms. Furthermore, our experiences made in high-bandwidth network testing made it possible for us to satisfy the requirements for our tests of both the high-bandwidth IDS tests and tests of the DDoS mitigation framework, such as throughput, automation capabilities, and reliability of the test environment.



III

ACCELERATION OF INTRUSION DETECTION SYSTEMS

6

INTRODUCTION TO INTRUSION DETECTION SYSTEMS

Essentially, there are two basic Intrusion Detection System models. For one, there are host-based Intrusion Detection Systems on personal computers or servers. They analyze data on the machine they are installed on, for example, log files to find unauthorized access to the system or they actively find and fight malware on the system. In the second case, they are usually called anti-virus or malware detection systems. Additionally, there are network-based Intrusion Detection Systems (NIDS) analyzing the network based on flow metadata on the one hand and the full network data on the other. Going forward, as this work is network-centric, this thesis will focus only on the network-based mechanisms. Network-based IDS are often used in central locations in backbone networks where the data rate is the highest.

NIDS use two fundamentally different attack detection mechanisms: Signature-based mechanisms and anomaly-based mechanisms, which will be explained in the following.

6.1 SIGNATURE-BASED NIDS

Signature-based NIDS compare the network traffic with signatures of previously known attacks. These systems are very reliable in detecting known attacks and attack vectors, especially of automated attacks. However, they are not capable of finding new attack methods such as zero-day exploits.

Hereby, signatures in the system can be represented in a couple of different ways [54].

STATE-MODELING The signature is modeled through states and state transitions. Here, only one specific order of state transitions can be covered, and if a state has not been recognized, the entire attack is not recognized. Alternatively, Petri nets can also be used to map a more complicated tree structure.

EXPERT SYSTEMS Systems that detect intrusions based on rules that describe the intrusion of attackers into the network are called expert systems. Often techniques such as forward chaining are used for the description. Expert systems are particularly suitable if new data are to be regularly included in the system. These systems are very

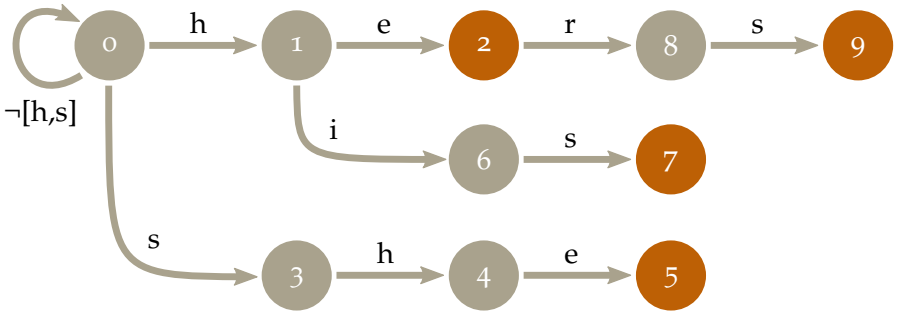


Figure 16: Example goto function used in Aho and Corasick [45].

i	1	2	3	4	5	6	7	8	9
f(i)	0	0	0	1	2	0	3	0	3

Table 5: Failure function for the example pattern matching machine used in Aho and Corasick [45].

flexible and powerful, but also slower than simpler models. Simple rule-based systems are a simplified version of these expert systems.

STRING MATCHING Searching specific symbol sequences in the packages is called String Matching. String Matching is easy to implement but very inflexible. The difficulty here is to find strings even if they are distributed over multiple packets, especially if the packets are not routed through the IDS in the correct order. In addition, even small deviations in the character strings cause an attack not to be detected. Modern IDS contain mechanisms to calculate distances of signatures to the strings saved in the ruleset of the IDS. String matching in IDS is very often based on the Aho-Corasick Algorithm [45]. Furthermore, NIDS make use of regular expressions to offer more flexibility to the detection mechanism.

6.1.1 *Aho-Corasick Algorithm*

The Aho-Corasick Algorithm is a string-matching algorithm first published in 1975 by Alfred Aho and Margaret Corasick [45]. Nowadays, it builds the foundation of most string matching systems. The algorithm is designed to find a finite number of keywords in a string of text as efficiently as possible, i. e., in a single pass through the text. For this, a pattern-matching machine must be constructed first, an example of which can be found in Figure 16. This example was also used in the original publication. It shows a machine matching the set of keywords {he, she, his, hers}. The graph is called *goto function*. State 0 is the start state, any symbol other than h or s leads back to state 0; for h or s, the algorithm follows the respective edge. If there is no state transition edge for the incoming symbol (such as for the

i	2	5	7	9
output(i)	{he}	{she, he}	{his}	{hers}

Table 6: Output function for the example pattern matching machine used in Aho and Corasick [45].

input string *ha* in state 1), the algorithm follows the failure function in Table 5 mapping a state into another state. The orange states (2, 5, 7, and 9) are states that return a match of one or more keyword according to the output function (Table 6).

The construction of the matching machine begins with the construction of a *goto graph*. Every graph begins with the start state 0; then, all keywords are added one by one to the graph in arbitrary order. New edges are only added if necessary. The keyword is added to the output function of the state at which its path terminates. When all keywords are added to the graph, the loop is added to state 0 for all input symbols other than those on the outgoing edges. The failure function is the last part missing to finish the matching machine. Given the depth of a state as the number of edges between the state and the starting state 0, we begin to build the failure function at the states with the lowest depth and then go up until we reach the states with the deepest depth. For $\mathrm{depth} = 1$, the failure function is always state 0. Now, to find the failure function of any state s with depth $d > 1$, we take the nonfail values of all states of depth $d - 1$ (incoming edges of these states). If one of the incoming edges has the same value as the incoming edge of s, this state can be set as target of the failure function for s.

This algorithm allows for very fast string matching. Modern IDS such as Snort make use of this algorithm by building one or a small group of state machines for the combination of all string matching rules. Unfortunately, the algorithm only works for strings and not for regular expressions. For those, IDS still need one state machine for each expression.

6.2 ANOMALY-BASED NIDS

Anomaly-based systems monitor network traffic and attempt to report suspicious behavior that can indicate an attack. The great difficulty for these systems is to reduce the false-positive rate to a practical level. Since in general only a tiny part of the network traffic belongs to attacks, even a false-positive rate of say 1% — which appears to be low at first glance — can make an IDS mostly unusable. As a thought experiment, assuming 0.001% of the network traffic would be attacks and attacks would be reliably detected, if only 1% of the network traffic is detected as false positive, only every thousandth attack alert is a real attack.

Anomaly-based Intrusion Detection Systems can be divided into different subgroups. The following subdivision follows the classification of Garcia et al. [128]. Dividing the systems according to their techniques, three classes, namely statistic-based models, knowledge-based models, and machine learning, can be identified.

STATISTIC-BASED MODELS In a learning phase, the statistical models provide for the creation of a network profile based on recorded network traffic. The metrics for this profile are, for example, the transmission rate, the number of packets of a protocol, the connection rate, or the number of different IP addresses. If the IDS is now in operation, i. e., in the recognition phase, a profile of the current network traffic is created and compared with the previously created profile. Deviations above a certain threshold value are signaled as an alarm.

The first statistical methods used univariate models with independent Gaussian random variables [108]. Later, multivariate systems with dependencies between the random variables were introduced [335]. Further approaches are the use of empirical distribution functions, Markov models to map TCP connections or the modeling of the connection with gray value matrices [83]. Another method derived from information theory is measuring the entropy of regular network traffic and comparing it with current network traffic. The idea is that attacks have a different redundancy than ordinary network traffic [182].

A big advantage of statistical methods is that no prior knowledge about network traffic is required. Instead, the knowledge can be obtained directly by observing network traffic. However, not every attack scenario can be detected with statistical data as they do not influence the network statistics significantly (e. g., small, targeted attacks).

KNOWLEDGE-BASED MODELS Expert systems are also available for anomaly-based NIDS (A-NIDS). These represent one of the most frequently used types of detection for A-NIDS. They classify the audit data in three steps. First, different characteristics and classes of data in the training data are identified; then, rules, parameters, or procedures are derived. The audit data is then assigned to the corresponding rules, parameters, and procedures.

Specification-based anomaly detection is a model in which human experts create the model based on the specified system behavior. With a complete specification — and if legitimate behavior beyond the specified can be excluded — this method is very effective. Only attacks that follow the specification cannot be detected by such a system. However, it is problematic that many network systems — especially those considered in this thesis — do not allow a simple limitation to clear specifications. Especially when considering Internet traffic, not

adhering to specification is quite common. Another problem with these systems is their low flexibility. In order to extend the system, the specification, implementation, and IDS must be adapted. The more the specification is extended, the higher the risk that attacks within the specification become possible. For example, specification-based anomaly detection systems in downward-compatible systems can hardly detect downgrade attacks.

Specifications for these systems can be defined in a few different ways. For one, in a finite automaton, in a description language [257, 319, 325], or with a rule-based classification.

MODELS BASED ON MACHINE LEARNING Machine learning creates knowledge based on accumulated experience of a technical system. In a learning phase, such a system gets to know the environment itself and thus adapt to the special conditions of this environment. Changes in the system then require a new learning phase, but possibly no changes to the IDS itself. Practical problems arise from the fact that training data must be created that is guaranteed not to contain attacks and represents the real network traffic as well as possible. However, one advantage of these technologies is that learning during operation is basically possible, but there is also the danger that an attacker can slowly change the network traffic training the IDS accustomed to the changed state and thus remain undetected. Unlike statistics-based models that often require a learning phase, the model itself is not necessarily known in advance. While statistics-based models learn parameters for their system, machine learning approaches build and refine the model itself during the learning phase.

There are many different approaches to machine learning. The best known and most common methods in the context of NIDS are explained in the following.

- *Bayesian networks* represent probabilistic relationships between variables by an acyclic graph. The nodes represent random variables and the edges conditional dependencies [327].

- *Markov models* can be divided into Markov chains and Hidden Markov models. Markov chains are state machines in which probabilities are assigned to the transitions. These can be used to model the topology and capabilities of the system. The Hidden Markov Model sees the system as a Markov chain, but its properties are not observable. There are numerous IDS models based on Markov models [204, 336, 337].

- *Artificial neural networks* — used for example, in an IDS by Mikkulainen et al. [265] — simulate the approach of biological neural networks in the human brain. Neural networks are very flexible and can adapt to changes in the environment compara-

tively easily. Neural networks, however, often make it hard to reconstruct the thought process behind decisions being made.

- *Fuzzy logic* is a generalization of Boolean logic by blurring truth statements. In IDS, fuzzy logic can be used to model network traffic, since its properties can be easily mapped [73]. A behavior is considered normal if it moves within a certain range. Fuzzy logic is often used here in connection with data mining techniques [113]
- *Genetic algorithms* are a special form of evolutionary algorithms that use various evolutionary biology-inspired techniques such as inheritance, mutation, selection, and recombination. In IDS, genetic algorithms can be used to create classifications [194], to find various parameters and properties of detection processes [73], or to filter network traffic [149].
- *Clustering techniques* assign the observed data to groups. These groups are often represented by a representative point in this group. Data points are then assigned a distance to these groups, such as the *Euclidean distance* or the *Mahalanobis distance*. If the distance exceeds a certain threshold, an anomaly is detected. Mananadhar and Aung developed an IDS system on this basis, which is limited to TCP headers and other easily accessible data [207].

6.3 OVERVIEW OF AVAILABLE IDS SYSTEMS

Table 7 gives an overview of different IDS systems that — according to the manufacturer — also work anomaly-based adapted. The table was adapted from Garcia et al. [128]. Three open-source solutions are especially noteworthy, as they are widely used in research. Snort [263] is a signature-based IDS, but with extensions. it also offers rudimentary anomaly-based detection capabilities, such as the SPADE (Statistical Packet Anomaly Detection Engine) plug-in, which has been discontinued. Snort is available in two different versions. Snort 2 is the current stable version, while there is also a preview version of the upcoming Snort 3. Snort 3 is a completely rewritten version. Unlike Snort 2, Snort 3 features multithreading. In 2013, Cisco took over the development of Snort. The Zeek Network Security Monitor [247] (formally known as Bro Network Security Monitor) is a framework for Intrusion Detection Systems that comes with its own scripting language. While Zeek can be used as an NIDS, it also can analyze network events. Both Snort and Zeek are based on the libpcap library. The third open-source platform is Suricata, developed by the Open Information Security Foundation [300]. Suricata is closely related to Snort. For instance, rulesets written for Snort can also be read by Suricata; some third-party tools are also compatible.

Name	Manufacturer	H	R	Anomaly-related techniques
AirDefense Guard	AirDefense & Inc.	•	•	Context-aware detection, correlation and multi-dimensional detection engines
Barbedwire IDS Softblade	BarbedWire Technologies	•	•	Protocol analysis, pattern matching
BreachGate WebDefend	Breach security	•		Behaviour-based analysis, statistical analysis, correlation
Zeek (Bro)	Lawrence Berkeley National Laboratory	•	•	Application level semantics, event analysis, pattern matching, protocol analysis
Checkpoint IPS-1	NFR Security	•	•	Confidence indexing
Cisco Intrusion Prevention System	Cisco Systems	•	•	Behaviour analysis, statistical analysis
DeepNines BBX Intrusion Prevention (IPS)	DeepNines Technologies	•		Multi-Method Inspection (MMI), behaviour analysis, protocol analysis, data correlation
EMERALD	SRI	•	•	Rule-based inference, Bayesian inference
FireProof	Radware Ltd.	•		Protocol anomalies
Firestorm NIDS	Gianni Tedesco	•		Protocol anomalies
Mazu Profiler	Mazu Networks & Inc.			Behaviour analysis (heuristics)
ModSecurity	Ivan Ristic	•		Event correlation
Network at Guard (NG)	C-DAC (formerly National Centre for Software Technology)	•	•	Protocol anomaly detection, statistical analysis
Next Generation Intrusion Detection Expert System (NIDES)	SRI	•		Statistical analysis
Nitro Security IPS	Nitro Security	•		Behaviour analysis
nPatrol	nSecure			Statistical analysis (profiles)
Portus (PAD)	Livermore Software Laboratories, Inc.	•	•	Protocol anomaly detection
Prelude IDS	Yoann Vandoorselaere et al.			Open platform/multiple anomaly-based modules available (3rd party)
SecureNet IDS/IPS	Intrusion Inc.	•	•	Protocol decoding, protocol anomalies
Siren	Penta Security	•	•	Abnormal user behaviour
Snort IDS	Marty Roesch	•		Open platform/multiple anomaly-based modules available (3rd party)
Snort_inline	Rob McMillen	•	•	Open platform/multiple anomaly-based modules available (3rd party)
Sourcefire ETM	Sourcefire Inc.	•	•	Network behaviour analysis
SPADE	Silicon Defense			Statistical analysis
StealthWatch	Lancope	•	•	Network behaviour analysis, "concern index"
Strata Guard IDS/IPS	StillSecure	•	•	Behaviour analysis, protocol anomalies
Symantec Intrusion Protection	Symantec	•	•	Behaviour-based
TippingPoint Intrusion Prevention System	3COM/TippingPoint Technologies	•		Statistical analysis, profiles
Toplayer Attack Mitigator IPS	Top Layer Networks	•	•	Statistical analysis, profiles

Table 7: Overview over anomaly-based NIDS systems from Garcia et al. [128]. H = Hybrid, R = Response

It is noticeable that the anomaly-based systems are all based on a signature-based detection model. Most systems, therefore, rely on a hybrid solution in order to be able to use the advantages of both models. Some of the systems provide active responses to threats, for example, by integrating a firewall, resetting TCP connections, or inserting a honeypot. Most manufacturers do not make any statements on how extensive anomaly-based detection is used in their systems or what techniques are used [128].

6.4 CIRCUMVENTING INTRUSION DETECTION SYSTEMS

To bypass IDS systems, various techniques are used, which were compiled by Chaboya et al. [88]. For example, it can be exploited that the IDS may handle network traffic differently than the target system since the IDS often uses a different operating system than the target and is often located in a different part of the network. The IDS can also be overloaded, which may cause the attack packets not to be analyzed by the IDS, or the IDS might be drowned in a flood of attack messages. IDS stimulators are used to trigger as many alarms as possible to achieve exactly that. Furthermore, attacks can be masked. For example, the target system may interpret `0x2f` as `/`, but the IDS may not. This can then allow a directory traversal attack past the NIDS. Zero-day attacks are impossible to detect for signature-based systems. Anomaly-based systems, in principle, can detect even zero-day attacks; however, as they were never tested against these attacks, there can be no guarantee.

To avoid detection by signature-based systems using pattern-matching, polymorphisms can be used, i. e., the malicious code is changed regularly, for example, by encrypting it each time with a different encryption algorithm [110].

Another method to circumvent detection is to distribute the content of packets so that an IDS must first receive and assemble the entire packet stream to analyze the content [97].

The analysis of IDS alerts is usually done manually by administrators. One way to ensure that an attack detected by the IDS is still successful is to make the administrator believe that the attack was unsuccessful. This can be achieved, for example, by not attempting to connect to the system immediately after detecting a backdoor in a system. Thus the scanning of the network is recognizable; the success in finding a gap is not.

6.5 STATE OF THE ART IN IDS ACCELERATION

As NIDS need to analyze and keep up with all traffic passing through entirely, the problem of throughput rising faster in networks than processing power has affected IDS since their inception. Many ap-

proaches towards reducing the impact have been analyzed. The approaches can be roughly summarized in two categories [84]. On the one hand the selective omission of data (data reduction) in order to control the data rate either randomly (sampling) or more intelligently by utilizing known features of the network and on the other hand the distribution of the network stream to several instances of the IDS and thus parallelization of the processing.

6.5.1 *Data Reduction*

Data reduction can be achieved by sample analysis instead of a complete analysis of all network traffic. Brauckhoff et al. examined the effects of the missing data and came to the conclusion that no negative effect is detectable with volume-based analysis methods, but flow count metrics are affected [69]. Mai et al. reviewed several sampling procedures and found impairments of anomaly-based IDS, both in terms of false-positive and false-negative rates [205], especially in the detection of portscans [206]. Contrary to the classical sample analysis, the current data filtering algorithms do not proceed randomly but consider specific properties of the network traffic. Sample analysis taking IP flows into account significantly reduces the error caused by data filtering. In these algorithms, samples are taken from short connections rather than from long ones [237]. Another method to improve sample analysis is to ensure that the maximum number of samples the system can handle is taken. Such a method was presented by Braun et al. [71].

The heavy-tail property of TCP connections can also be used for data filtering. Heavy-tail here means that only the first packets of a connection are essential for security analysis. If, for example, it is made clear after an SSH connection has been established that the user has authorization for this connection, further packets in this connection can be left out of the analysis without hesitation and, for example, a possibly extensive backup copy does not have to be completely analyzed by the IDS. However, attackers knowing how the IDS analyzes the traffic could move attack traffic to the end of the payload to circumvent the IDS. A pre-filter that allows this pre-sorting has been implemented on an FPGA platform [320] and connected to the Bro — now Zeek — IDS [134].

6.5.2 *Parallelization*

Parallelization is the second approach that has found widespread use. Vasiliadis et al. [307], for example, rely on GPU's unique capabilities for parallel data processing with their development based on the open-source program Snort. They doubled the throughput compared to the unmodified version of Snort. They moved the regular expres-

sion parsing from the CPU to the GPU and therefore moved one of the most computationally stressing parts of the data analysis off the CPU. The high rate of acceleration of the system is elevated by the fact that this older version of Snort only utilizes one CPU core. No attempt has been undertaken to build a similar system with the new, upcoming version of Snort under version number 3. Snort 3 is a complete multicore re-implementation of Snort.

The open-source IDS Suricata [300] also features GPU-acceleration in some older versions. However, the development has been discontinued, and the feature was removed in 2018 as the feature was *"unmaintained, untested and very likely broken"* [291].

Load distribution to several computers is also a widespread approach. The Zeek Network Security Monitor [301], for example, is designed to be distributed on multiple machines in the network. In addition to a central instance coordinating the IDS, several worker instances analyze network traffic. Difficulties to be considered with such a system were described by Vallentin et al. [305] and solved for Zeek [245]. First, the traffic must be evenly distributed among all analysis nodes to minimize the necessary communication between instances. In addition, the nodes must coordinate their analysis at the lowest level, and the results must be validated. It is also possible to run multiple Zeek instances on one computer. This way, the computer's entire computing power can still be fully utilized, even though each worker uses only one CPU core. The network card then distributes the different connection streams to the different instances [245]. How a data stream can be distributed to different instances was described by Schneider et al. [275]. In 2003, Cisco patented a method for distributing network traffic from a load balancer to many instances and finally analyzing it in parallel signature-based IDS instances [279].

FPGAs are very well suited for parallel data processing; therefore, approaches are also being pursued to implement parts of the IDS in them. Jiang et al. have implemented rules of the Snort IDS on a Xilinx-5 [158]. The packet classification (and the analysis of Internet traffic [171]) can be done on an FPGA [284]. Pontarelli et al. follow the approach of splitting parts of the rules into several FPGAs and have specialized hardware implementations available that can implement this part of the rules particularly fast [250].

In order to parallelize statistic-based NIDS systems, it is necessary to be able to distribute the statistical methods. Such a model was developed by Amann et al. [51].

Parallel processing can only be used if different actions are not interdependent, i. e., they do not need to be processed sequentially. The string matching algorithms used by the IDS systems for recognition are strictly sequential and work through each byte one by one in the naive implementation since they work with finite automata. One way

to speed up parsing is to parse several bytes simultaneously. Without knowing the previous state of the machine, however, it is difficult to do this. Luchaup et al. [200], therefore, propose to estimate a state and later correct the results if necessary if the estimate turns out to be wrong. The estimates are supported by statistical analyses to achieve the highest possible number of hits [200].

A review of literature in this research area revealed several previous attempts in combining IDSs with FPGA-based hardware prefilters. The majority of academic prototypes are built with a NetFPGA 1 card, an FPGA-based board with network interface card functionality. One such prototype is the Shunt project [320] later combined with the Zeek IDS [134], this architecture offloads decisions based on IP addresses and connection tuples onto a NetFPGA pre-filter. Zeek manages a shared memory between itself and the Shunt to send instructions. Zeek is a comprehensive network security monitor with an extensive set of features [246, 306]. It provides a framework that includes a scripting language; this allows deployers to tailor Zeek to their specific needs. A different approach was proposed by Song et al. [285]. Their idea was to add a string-matching hardware prefilter to the signature-based Snort 2 IDS. They implemented string matching in an FPGA via Bloom filters, which still allows for false positives. Nonetheless, Song et al. show that the amount of packets that still have to be analyzed by the Snort IDS is massively reduced. Snort [263] is popular both in practical applications and in research featuring rulesets that are often used by researchers to test their own and other implementations (e. g., [64, 100, 151, 332]). Floyd et al. [121] translated a regular expression into integrated circuits. Translation of generic regular expressions was done soon after [282] to accelerate the command-line tool *grep*. Clark et al. [100] provide a methodology to implement regular expressions using an NFA in FPGAs, specifically for IDSs. This provides a rudimentary framework and guidelines for future implementations. Hutchings et al. [151] created a conversion of regular expressions to an HDL. This resulted in a 600x increased performance compared to software. Bispo et al. [64] set out to create an efficient implementation of regular expressions on FPGAs. They provide techniques to reduce spatial cost and to increase performance, improving efficiency by 10x. Lin et al. [197] investigated the efficient implementation of regular expressions on FPGAs. They propose the extraction of common sub-regular expressions and share these in order to achieve a reduced area cost. They did not evaluate their system concerning throughput. Baker et al. [57] created a custom microcontroller on an FPGA, that is able to interpret DFAs. Furthermore, other research has also targeted efficient implementations of DFAs in FPGAs [74, 225]. Yang et al. [333] provide a conversion from regular expressions to an HDL and also propose several optimizations, such as the multi-character input matching optimization, similar to [332].

Jaic et al. [155] combined Snort with an FPGA to offload regular expressions for Solarflare AOE devices. Researchers have also increased the throughput of firewalls by combining them with FPGAs, for instance, Hager et al. [139, 140], and Fießler et al. [120].

While many attempts at accelerating IDS with FPGAs were undertaken, most of these systems fail to report how many rules are possible to be implemented and the space requirements on the FPGA or built their systems solely in simulators. A full system implementation capable of handling regular expressions on FPGAs integrated fully with an IDS cannot be found in the literature.

7 PROBLEM STATEMENT

Intrusion Detection Systems (IDS) are important for security in networks to find and circumvent attacks. Recent attacks (e. g., botnet software such as Mirai [180]) show how legacy systems in networks can be a liability. Intrusion detection can help to mitigate the extent of these attacks as they can function as an early warning system. However, network bandwidth increases fast; it increases a lot faster than computing capabilities of hardware platforms [131, 224]. At the same time, detection mechanisms become more and more complex. From simple string matching to more complex regular expression matching to metadata analysis and sophisticated machine learning algorithms for anomaly detection, intrusion detection can become quite resource-demanding. The core of most modern IDS is the string matching and regular expression matching engine using 75% of the total CPU processing time [52, 79, 307]. It is a very resource-intensive part of the system and therefore offers a big opportunity for improvement. CPUs are designed to offer medium performance for any generic calculation. However, specialized processors can be tailored to the use case of their operational area and optimized accordingly. A design of a use case-specific processor can, therefore, in theory, improve the matching capabilities of IDS. In the following, we want to discuss three basic concepts for such processors. For one, we look into an FPGA-based system where the regular expressions themselves are translated and molded into hardware. The second design is an FPGA-based co-processor with its own regular expression based assembly language that can be translated into an ASIC. Furthermore, we will look into graphics processors (GPUs). GPU cores are much smaller and less versatile than CPU cores, but for specific applications that share key features of graphics processing, moving the calculation from the CPU to the GPU often enables far-reaching improvements.

Parallelization has proven to be a viable solution for many resource-hungry algorithms in computer science. Today's multi-core architectures for CPUs make parallel computing possible even on commodity hardware. However, multi-purpose processors lack the optimization of specialized hardware in the form of ASICs or FPGAs or the massive parallelization options of graphics processors. We want to analyze whether Intrusion detection systems could benefit from specialized hardware modules. There are two fundamentally different architectures for such systems:

- The pre-filter approach consists of a system that sits in front of the IDS that pre-sorts the data. This means that the pre-

filter assesses whether the packets are benign or require a more detailed analysis. Such a filter does not have to detect attack traffic perfectly but has to be sure about benign traffic.

- A co-processor is a highly specialized processing unit to which parts of the processing workload can be delegated. Unlike a pre-filter, a co-processor must be able to analyze the data perfectly and distinguish between legitimate and harmful traffic.

We identified three different hardware options that can be used for the two models.

- For one, a Field Programmable Gate Array (FPGA). An FPGA is an integrated circuit that can be programmed in the field, i. e., while in service. The reprogrammability of an FPGA gives the unique possibility to bypass the restrictions of fixed transistor circuits. Instead of instructions that have to be interpreted and then sequentially processed, the instructions can be translated directly into circuits. This is particularly useful for the pre-filter approach; the implementation of the regular expressions matcher can be done directly in the wiring of the transistors.

- Another approach is the Application Specific Integrated Circuit (ASIC). Unlike a CPU, an ASIC is not able to execute arbitrary instructions but is restricted to very few, more complicated instructions — if it supports instructions and is programmable at all. This limitation of the task area makes it possible to optimize the hardware much further. Typical areas of application for ASICs include fast Fourier transformations in digital signal processing or the voice digitization in older mobile phones. It has to be investigated how such a co-processor as ASIC could be implemented.

- The third approach is the Graphics Processing Unit (GPU). A GPU is intended to manipulate and alter memory for image creation to output these images on a display device. The GPU should display these images in real-time with as little delay as possible at high frequencies. A GPU features many small processing cores while a CPU features very few, bigger cores. In contrast to a CPU, a GPU is highly optimized for manipulating large matrices and vectors in parallel. In recent years, the GPUs' potential as a Co-Processor for other uses than those directly related to image manipulation has been recognized. GPUs have often been used to mine cryptocurrencies and for machine learning. Vasiliadis et al. [307] already saw the potential to use GPUs as a Co-Processor for IDS back in 2008 and the Suricata IDS offered support for GPUs until recently.

7.1 RESEARCH QUESTIONS

Based on the models and hardware options in the aforementioned deliberations, we identified the following research questions:

1. Based on maintainability, cost, and acceleration potential, which of the hardware options is the most promising?
2. Based on the decision following the first question, how can this option be implemented?
3. Can this option accelerate the IDS sufficiently?

In order to obtain the necessary results to answer these questions, several steps are necessary:

- An analysis of how prototypes of the three different hardware-based acceleration mechanisms can look like.
- A prototype of the mechanism that shows the most potential.

8 HARDWARE-BASED IDS ACCELERATION SYSTEM

Network-based Intrusion Detection Systems rely heavily on the performance of its string matching and regular expression matching engines. String matching is done with the highly performant Aho-Corasick algorithm. Regular expressions are converted to finite automata through a series of algorithms. These finite automata, in turn, are applied on the packet stream to analyze the traffic.

CPUs are generalized processing units capable of every potential processing operation possible. However, for specific purposes, other processing designs can be a lot more promising. Therefore, we analyze how specialized hardware can be used to accelerate Intrusion Detection Systems.

We analyze several aspects of this approach. For one, we need to assess how regular expression matching works in software to see how this can be adapted to hardware. Then, the location of the hardware module needs to be assessed. Two basic designs are possible: A prefilter analyzing the traffic before forwarding it to the software-based IDS and a Co-Processor, which is called by the CPU for help when necessary. For the second design, it is also necessary to optimize the packet forwarding process to eliminate possible bottlenecks. The traffic does not have to be analyzed by the CPU; therefore, options to bypass the CPU need to be explored.

Hardware-based acceleration options require the purchase of this very hardware. It needs to be assessed whether the acceleration achieved sufficiently offsets the costs of this purpose-built or bought hardware compared to the cost of a basic cluster-based approach using the default software implementations.

While none of the common open-source IDS (Snort, Suricata, and Zeek) currently support any of the hardware-based acceleration methods, at least Suricata did support GPU-acceleration in an older version. Therefore, Suricata's implementation also needs to be assessed.

As a basis for our implementations and evaluations, we choose the Snort IDS due to its prevalence, modern implementation (complete rewrite for version 3), and well-supported and regularly updated community ruleset.

8.1 REGULAR EXPRESSIONS AND FINITE AUTOMATA

Regular expressions are a description language for search patterns. A regular expression processor or regex processor interprets the regular expression and applies the regular expression on text to find the

Dominik Lang [26] has contributed to this section with his master thesis.

pattern described in the expression resulting in a match or no match and a position or several positions of the match. Regular expressions consist of *meta characters* and *regular characters*. Regular characters are characters to be matched. A regular expression only consisting of regular characters is a simple string matching mechanism. The metacharacters are instructions for the regex processor on how to interpret the regular characters. It can give the processor the instruction to not only match one occurrence of the regular character but many (by giving it a specific amount or a range which can go from zero to infinity) or invert the characters listed (match if this character is *not* present). The wildcard matches for any character and can be used, for instance, when a pattern should match specific keywords at the beginning and the end of a string while the characters in between do not matter.

In the Chomsky hierarchy, regular expressions are a description language for the type-3 grammar or *regular grammar*. This means they have the same expressiveness as finite automata. This fact is widely used for regex processors for the internal representation of the regular expression and the implementation of the state machine matching the expression on the text. Finite automata are usually represented either as non-deterministic finite automata (NFA) or deterministic finite automata (DFA). They contain states and state transitions that are dependent on the input. The fundamental difference between NFA and DFA is that an NFA can contain several state transitions from one state to different other possible states on the same input. DFAs are a subgroup of NFAs where the amount of such non-deterministic transitions is zero. Implementing an NFA requires the implementation of *backtracking* where, when one path through the state machine leads to a dead end, the machine goes back to the state of the last non-deterministic transition and tries a different way. Only when all possibilities were tried or the pattern matches, the machine stops. While DFAs are merely a subgroup of NFAs, there still is a DFA for every NFA; i. e., every NFA can be transformed into a DFA. NFAs require less memory than their DFA counterparts; converting an NFA to a DFA can, in the worst case, lead to an exponential increase of states (n states in an NFA could lead to 2^n states in a DFA). However, no backtracking is needed, which leads to far better performance on sequential machines such as CPUs.

Despite the potential exponentially higher space requirements for DFAs, modern Intrusion Detection Systems such as Snort still manage their rulesets as DFAs as the amount of memory used is still manageable for modern computers and the advantages of not needing to implement backtracking and the speed advantage far outweigh the space requirements.

As the naïve, general conversion of regular expressions to a finite automata leads to non-deterministic finite automata, a general trans-

formation from NFA to DFA is a necessary second step for the internal representation of regular expressions in memory in most IDS.

8.1.1 *Thompson's Construction Algorithm — Regular Expression to NFA Conversion*

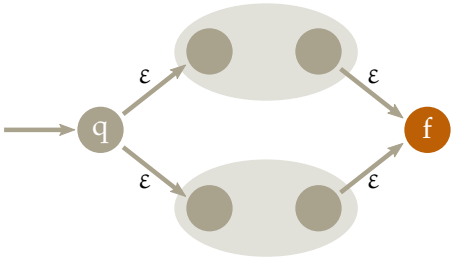
The transformation of regular expressions to NFAs is a topic that has already been worked on quite a while ago. McNaughton and Yamada [215] and Thompson [302] developed the algorithms in the 1960s that are still used today. The algorithm is, therefore, often called the McNaughton-Yamada-Thompson algorithm or Thompson's construction algorithm. In principle, the algorithms follow a recursive divide-and-conquer concept. Thompson creates sub-expressions and converts them to sub-graphs, combining them to build the full expression according to a set of rules described in the following with the graphical representation based on Aho et al. [46, Chapter 3.7.4], with the end state in orange.

The empty-expression ε shall be represented with:

where ε represents an empty transition.

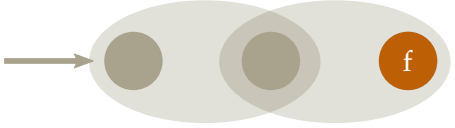
Implementing a symbol (i. e., regular character) a is done with the expression:

The union expression "|" ("or") is represented with



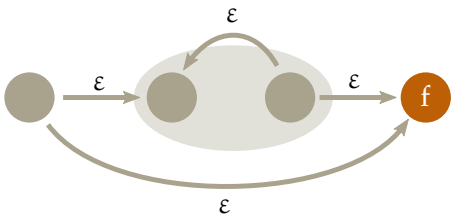
where the gray ellipses represent sub-graphs that need to be included here when necessary. The connections go to the initial state of the sub-graphs and emerge at their final state.

For the concatenation, where two sub-expressions follow each other in the regular expression without any symbol in between, the graph looks as follows



where the final state of the first (left) sub-graph merges with the initial state of the second (right) sub-graph and becomes one.

The *Kleene star* expression "*" is represented by



where the ellipse is the sub-graph which the Kleene star affects.

Finally, brackets "(□)" that are used to explicitly group sub-expressions in regular expressions also explicitly group sub-graphs in the NFA.

With this set of rules, it is possible to prove that any regular expression can be implemented in an NFA.

8.1.2 *NFA to DFA Conversion*

Algorithm 1 Removing all empty transitions from NFA.

```
for all nodes N_i ∈ NFA from the last to the first do
    for all outgoing transitions do
        if transition is empty then
            follow transition to next node N_j;
            add all outgoing transitions from N_j to N_i;
            if N_j is a final state then
                N_i becomes a final state;
            end if
            Remove the empty transition;
        end if
    end for
end for
remove unreachable states;
```

Two steps have to be undertaken to convert the NFA to a DFA. For one, the NFA resulting from the previous step include a large number of empty transitions, which is one cause of non-deterministic behavior. Algorithm 1 needs to be performed to remove these empty transitions.

Now, there are still non-deterministic state transitions, for example, due to the Kleene star. To fully convert the NFA to a DFA, the Rabin and Scott powerset construction algorithm can be used [254] that, in principle, works as can be seen in Algorithm 2. The resulting DFA can now be used by the IDS.

Algorithm 2 Powerset Construction Algorithm.

```
Create a new empty state machine DFA with a set of states;
Add initial state of NFA to DFA;
for all state s in NFA beginning with initial state do
    add state s to DFA
    for all output character c in s do
        create new state t by combining the set of states C that c
        leads to.
        if t ∉ DFA then
            add all transitions from all states in C to t
            if any state in C is a final state then
                t is a final state;
            end if
            add t to DFA
        end if
    end for
    add transition (c → t) to s
end for
```

8.1.3 *Character Classes and Their Negations*

In theory, these algorithms suffice to produce DFAs from any regular expression. However, in many circumstances, these algorithms are impractical. Regular expressions often feature character classes where not only one character but several can lead to a state transition in the DFA. With the aforementioned algorithms, these would be split up, and several transitions would be added to the graph. Especially for small character classes, this works fine and also fixes the potential issue when two or more character classes transition to different states while sharing common characters.

However, the larger the character class, the larger the NFA, and in turn, the larger the DFA. This can lead to an explosion of state transitions, especially with negated character classes, where a transition happens when no character in the list matches. These were not foreseen in Thompson's construction algorithm, which means that in order to use the algorithm, the regular expressions featuring negative character classes have to be converted into regular expressions compatible with Thompson's algorithm first. With negative character classes, this is possible by inverting the class. Instead of listing the characters that cannot match, we can always list the characters that do match instead, provided that the list of all possible characters is known. It is evident that this is not practical. If in the most extreme case, we just do not want to match one specific character, even if the system only allows the original 7-bit ASCII characters, the set would still go up from one character to 127 and in turn to 127 transitions, not even considering 8-bit extended ASCII or 32-bit Unicode UTF-8.

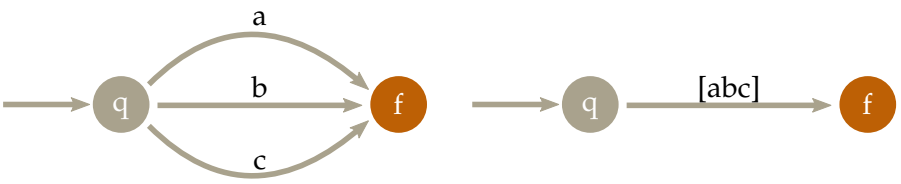


Figure 17: Difference between original Thompson's algorithm (left) and using character classes (right) for the same sub-expression.

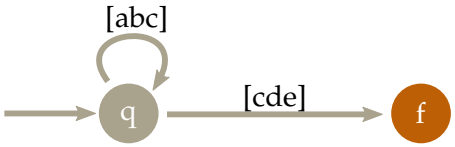


Figure 18: Example of an NFA with two transitions with overlapping character classes.

In some cases, the set of possible characters might even be unknown, as an attack might use invalid characters to throw of Intrusion Detection Systems. Therefore, the algorithms had to be adapted under the precondition that the full set of regular characters is unknown. For one, instead of breaking up character sets as the original Thompson's construction algorithm does, the algorithm keeps them as one state transition.

Figure 17 illustrates the difference. While on the left, the original Thompson's algorithm is used, the right uses character classes to condense three different transitions into one.

The removal of empty transitions remains unaffected by this; however, the powerset construction algorithm needs to be adapted to cope with several overlapping character class transitions. We are first taking a look into the most straightforward case of two overlapping transitions. There are three different possible cases:

1. Neither class is negated.
2. Both classes are negated.
3. Exactly one class is negated.

The transitions have to be split up in a way the powerset construction algorithm can work again. If neither class is negated, this can be very straight forward. Given the example in Figure 18 of two classes $A = \{a, b, c\}$ and $B = \{c, d, e\}$. To build the minimal set of transitions for the powerset construction algorithm to work, we need to observe the three different options for characters:

1. Character is in both classes.
2. Character is only in class A.
3. Character is only in class B.

Therefore, we build the following new classes:

$$\begin{aligned} A \cap B &= \{a, b, c\} \cap \{c, d, e\} = \{c\} \\ A \setminus B &= \{a, b, c\} \setminus \{c, d, e\} = \{a, b\} \\ B \setminus A &= \{c, d, e\} \setminus \{a, b, c\} = \{b, e\} \end{aligned} \tag{8}$$

With these new classes, the powerset construction algorithm works again.

Now, if both classes are negated, the deMorgan rule and $\overline{A} \setminus \overline{B} = B \setminus A$ (Theorem 2; Proof in Appendix) can help us build the new transitions as follows:

$$\begin{aligned} \overline{A} \cap \overline{B} &= \overline{A \cup B} = \overline{\{a, b, c\} \cup \{c, d, e\}} = \overline{\{a, b, c, d, e\}} \\ \overline{A} \setminus \overline{B} &= B \setminus A = \{b, e\} \\ \overline{B} \setminus \overline{A} &= A \setminus B = \{a, b\} \end{aligned} \tag{9}$$

If only one transition is negated, (without loss of generality, we choose to invert B) the following transitions can be built following directly from the properties of complements:

$$\begin{aligned} A \cap \overline{B} &= \overline{\overline{A} \cup B} = \overline{\overline{A \setminus B}} = A \setminus B = \{a, b, c\} \setminus \{c, d, e\} = \{a, b\} \\ A \setminus \overline{B} &= \overline{(\overline{A} \cup \overline{B})} = A \cap B = \{c\} \\ \overline{B} \setminus A &= \overline{(B \cup A)} = \overline{\{a, b, c, d, e\}} \end{aligned} \tag{10}$$

Furthermore, there is the special case of the *any* character in regular expressions ".". This character has to be treated as the negation of the empty set $\overline{\emptyset}$, and the aforementioned algorithms work as intended.

Generalized Case of More Than Two Transitions

To allow this algorithm to work with more than two transitions, we consider the case where we already built disjunct transitions for all other transitions, and another transition X needs to be considered. All transitions are already disjunct to each other except for X. Therefore, we only need to build disjunct transitions for each transition with X following the aforementioned rules.

Therefore, as we know we can build disjunct transitions for two transitions and we can build disjunct transitions for every newly added transition to a set of disjunct transitions, by complete induction, we can build disjunct transitions for any number of transitions.

8.2 CONCEPTS

Keeping this state machine generation in mind, three different types of hardware acceleration were investigated. The first one was to directly synthesize the regular expressions in an FPGA based on the

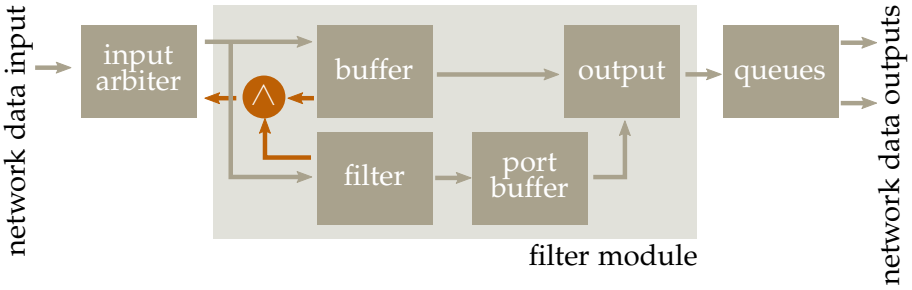


Figure 19: Data flow of packets through the modules of the NetFPGA; feedback line of the filter to stall the input in orange [10].

aforementioned state machines generated by the aforementioned algorithms. This model follows a pre-filter design, where the system filters out traffic that does not need further analysis and instead forwards any packet where a match occurred to an IDS. The second model follows a Co-Processor design, where the IDS is supported by a highly specialized processing core only capable of matching regular expressions in parallel. The Co-Processor design is then adapted to be used either with ASICs or GPUs. The GPU implementation requires the same transformation of regular expressions into state machines as the FPGA-based implementation.

8.2.1 *FPGA-based parsing of Regular Expressions*

The first model to realize a pre-filter for IDS is to mold the regular expression state machines directly into the hardware. As the data set of regular expressions changes over time — the snort database updates roughly daily — such a system can not be realized as ASICs. However, an FPGA module, which can be reprogrammed daily, can be used. As a first step, we build a system to match fields statically.

Matching Lower Layer Fields in FPGAs

Parts of this section have been published at IEEE FGCT 2014 [10].

The goal of the matching system is to provide a pre-filter for Intrusion Detection Systems. This system could, for example, filter flows in such a way that only flows to a specific subnet are analyzed or to certain ports. It could also function as a load balancer spreading the flows based on their subnet or ports over several IDS instances. As such, the throughput needs to be reliably high with minimal latency.

We used the NetFPGA platform[1] as the hardware base for our implementation. The basic concept of our filter module for the 10G NetFPGA platform is based on the work of Scott [277]. It consists of two parts. One is used for buffering and forwarding packets to the right output ports and the actual parser and filter. The path through the entire FPGA, including our filter module, is depicted in Figure 19.

1 https://netfpga.org

Packets are copied and take two parallel paths. The default output path at the top just forwards the packet to the original destination. The filter path at the bottom analyzes the packets and decides if a copy should be sent to the IDS. If yes, the packet now addressed to the IDS is also added to the output queue.

Our goals are aimed at a packet filter that provides maximal network throughput and minimal latency. This filter dynamically analyzes network headers and can forward the filtered packets to a more advanced filter, which can then perform arbitrary intrusion detection procedures or other processing tasks.

The parsing of the protocol stack is done in parallel while writing the input to the data buffer. This only interferes with the data flow if a header boundary lies within a word, as in that case, the flow is kept for an additional cycle. The process is implemented by a finite-state machine, in which each node corresponds to one layer in the protocol stack. First, the physical layer or MAC layer in the form of IEEE 802.3 has to be analyzed, if not to filter on that level, then at least to find the offset where the header of the next protocol starts. Filters based on MAC addresses are applied here. Next follows the IP layer, where filters based on the IP addresses are located, and in a third step, the transport layer protocols for port-based filtering are analyzed.

This implementation does not contain any payload data analysis. For that, the regular expression based filter extension is necessary.

The matching of static fields is fast enough to achieve line speed without any parallelization necessary.

Molding Regular Expressions in FPGAs

Dominik Lang [26] has contributed to this section with his master thesis.

Benjamin Schimmele [31] has contributed to this section with his diploma thesis.

Extending on this static approach, we built modules to filter based on regular expressions. This concept follows an asymmetric approach in the sense that the transformation and updating of regular expressions is slow, but the matching and execution is fast. For this use case, the slow transformation is unproblematic, as it is only performed when the rules themselves change without any real-time constraints. Similarly, this concept prioritizes speed when it comes to the time-memory trade-off by sacrificing resources and space on an FPGA in order to achieve higher throughput depicted in Figure 20.

The main idea is to translate the regular expressions into a hardware description language (HDL) (such as Verilog or VHDL), which is then directly synthesized onto the FPGA. We use the aforementioned transformation to build DFAs, based on which HDL code is generated. The DFA can then be directly converted to the HDL.In order to be able to match multiple regular expressions in parallel, state machines are generated, synthesized, and implemented for each regular expression. All state machines are then applied to every incoming packet in parallel. At the same time, the packet is also directly for-

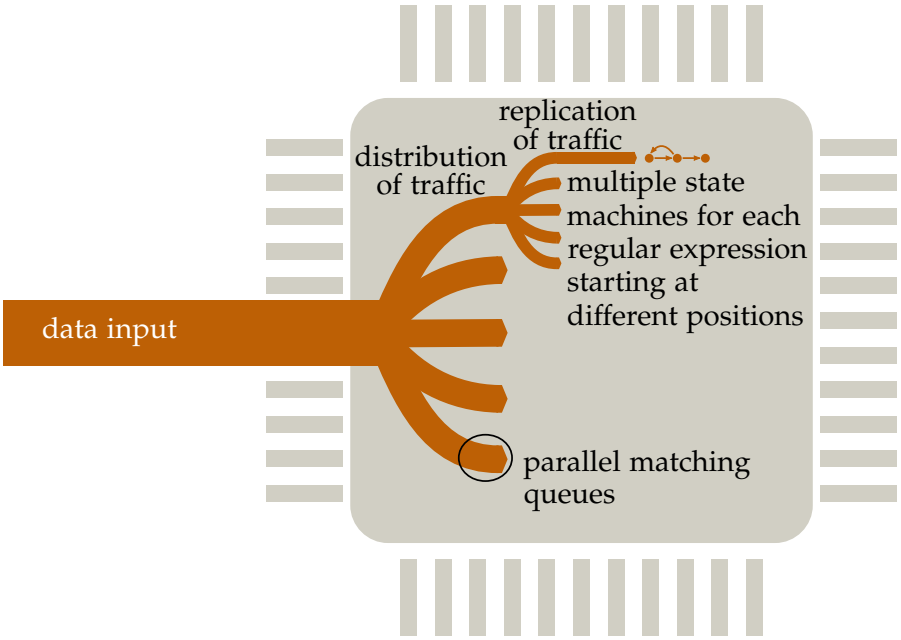


Figure 20: Concept of the parallelization model for the FPGA pre-filter approach. Based on [26].

warded to its original destination in order to not negatively influence regular networking behavior.

Only if one of the state machines reach a final state (i. e., the respective regular expression matches), the pre-filter will forward a packet to the IDS for further inspection.

In contrast to the static approach where we are matching fixed values at specific offsets in packets, much more of the traffic must be analyzed (including the payload) with much more complex analysis. A parallelization (distribution of the traffic) is, therefore, necessary to keep up with the packets coming in.

There are two challenges with this concept: matching at arbitrary positions and block size mismatch between the forwarding process and the matching process.

MATCHING AT ARBITRARY POSITIONS A challenge with this approach so far is to match patterns at arbitrary positions in a packet. The issue is that a regular state machine needs to compare bytes sequentially from start to finish and has a fixed starting position. Thus, in order to find patterns starting at arbitrary positions in a packet, a state machine needs to be started for every byte position in the packet. For this, the traffic needs to be replicated to be matched in multiple state machines in parallel. Consequently, in the worst case without any further optimizations, the number of state machines needed per regular expression is at least equal to the maximum byte length of a packet.

BLOCK SIZE MISMATCH A further challenge is potentially differing block sizes between the forwarding process and the matching process. In order to achieve line speed, the network interface card (NIC) implementation on an FPGA forwards the packets in block sizes of multiple bytes, for instance, 32 bytes per clock cycle. However, a regular state machine can only compare one byte at a time, resulting in a block size difference and consequently, a speed difference between the internal forwarding process and the state machines. The problem is that the state machines need to start matching at every position to recognize patterns at every possible position in a packet.

One approach to reducing the impact of this speed difference is to use pipelining. This means we start matching the next incoming packet while the matching process for the previous packet is still ongoing, but the forwarding process has already finished. However, all state machines then need to be replicated per queue.

Another approach is to modify the state machines to be able to match multiple bytes at once during a single clock cycle, i. e., transform the state machines into multi-character input matching state machines. Yang et al. [333] and Yamagaki et al. [332] provide different types of approaches to implement multi-character input matching.

In the ideal case, the state machine and the forwarding process are able to process the same amount of bytes per clock cycle. In that case, no further queues are needed to handle new packets. Otherwise, if the block size of the forwarding process is larger, then multiple queues are required; nevertheless, increasing the bytes that can be matched per cycle decreases the number of queues and the number of total state machines per queue needed. However, the more bytes a state machine is able to match per cycle, the more complicated its implementation becomes, resulting in a reduced clock frequency on the FPGA board. Thus, a good compromise needs to be found in order to gain the best performance.

Further optimization to reduce the number of queues needed is possible by adding an upper boundary on the maximum number of bytes a state machine matches before it returns with a default value of *success*. For example, a regular expression featuring "*.**" would only terminate at the end of a packet. By introducing a boundary of n bytes, the state machine terminates after a maximum of n cycles, potentially greatly reducing the number of queues. In addition, state machines can be reused for later offsets in a packet reducing the number of state machines required per queue.

It is possible to simplify the state machines by cutting off everything after a final state, as the IDS will analyze the traffic again regardless (i. e., pre-filter model).

PERFORMANCE ASSESSMENT Given the concept and the optimizations, it is possible to calculate the number of state machines and

queues required in order to estimate the space requirements on an FPGA.

The number of queues depends on the maximum number of cycles remaining to finish the matching process after a packet has been successfully forwarded. The remaining cycles for a state machine are either equal to its theoretical maximum number of cycles or until it reaches the end of a packet, whichever happens first. Thus, the number of queues q can be calculated with

$$q = \max_{c \in \{1,\dots,n_b\}}(\min(n_{rc}(c), rc(c))) \tag{11}$$

where c denotes the relative starting cycle of a state machine for a packet, n_b the maximum number of blocks per packet (which depends on the forwarding block size and the maximum packet size), n_{rc} the maximum number of remaining cycles for a state machine (i. e., until the specified upper boundary or the theoretical maximum number of cycles), and rc the number of remaining cycles for a state machine until it reaches the end of a packet.

Further, it is possible to calculate the number of state machines needed based on the number of queues. The total number of state machines depends mostly on the matching block size, the maximum number of matching cycles, and the number of queues (which depends on the former two). Hence, the number of state machines per queue (qsm) and the total number of state machines (tsm) can be calculated with:

$$\begin{aligned} qsm &= \left\lceil \frac{fbs}{mbs} \right\rceil \cdot \min(n_c, n_b) \\ tsm &= qsm \cdot q \end{aligned} \tag{12}$$

where fbs denotes the forwarding block size, mbs the matching block size, n_c the maximum number of cycles for a state machine (either the theoretical maximum or the upper boundary), and again n_b the maximum number of blocks per packet.

While tsm gives us the number of state machines per regular expression, we need to multiply this with the number of regular expressions we need to analyze to find the total number of state machines necessary for a full-scale system.

Realistic values for the parameters are

- forwarding block size fbs: 32 Bytes (based on the NIC reference implementation for the NetFPGA-Sume)
- matching block size: 1 byte
- maximum packet size equal to MTU: 1500 Bytes

- maximum number of cycles per state machine until returning *success*: 20 (number of queues needed for each regular expression is therefore 21)
- the community ruleset for Snort contains roughly 900 regular expressions.
- maximum clock frequency of 200 MHz.

We synthesized some regular expression engines on the NetFPGA-Sume to assess the space requirements. Unfortunately, given these parameters, to reach a line speed of 10 Gbit/s requires so much space on the FPGA that a maximum number of 10 regular expressions could be matched, which is significantly too small for the Snort community ruleset containing 900 regular expressions. This assessment does not include the space required for the NIC implementation itself.

In summary, the main advantage of FPGA regular expression matching is speed and flexibility; however, the main disadvantage is maintainability and the updating of the regular expressions. Using this approach, it is possible to optimize the regular expressions on a logic gate and spatial level to get the best performance and to use the FPGA resources in an ideal way. However, every time the regular expression set changes, the entire system needs to be rebuilt and deployed, which can take hours. Furthermore, the space requirements on the FPGA simply renders this approach unfeasible. The complexity of the state machines and consequentially low throughput per machine leads to a large number of state machines for each regular expression. With the high number of regular expressions needed, a cluster of many FPGAs would be necessary. The pre-filter design has its advantages as the CPU can be released from many processing demands by filtering out traffic before it even gets there. However, there are also many disadvantages. Having several locations where traffic is filtered leads to unclear responsibilities that require extensive documentation. Moreover, collecting metadata is severely hindered by this approach. Administration requires knowledge of several systems. Due to these issues, we abandoned this approach.

8.2.2 *Co-Processor Design*

Besides the pre-filter mechanism, there is also another possible design — the Co-Processor. In the Co-Processor design, the CPU still keeps control over the data and how it is processed. The main program running on the CPU can call the purpose-built Co-Processor for assistance only for operations where it promises better performance, while the CPU keeps full control over the whole process. The disadvantage is that a cautious implementation is necessary to prevent the CPU and the bus system from becoming a bottleneck.

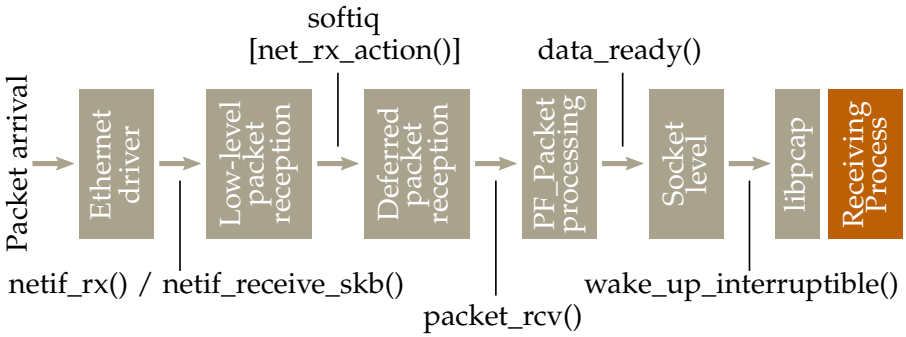


Figure 21: PF_PACKET processing pipeline on Linux (Braun et al. [72]).

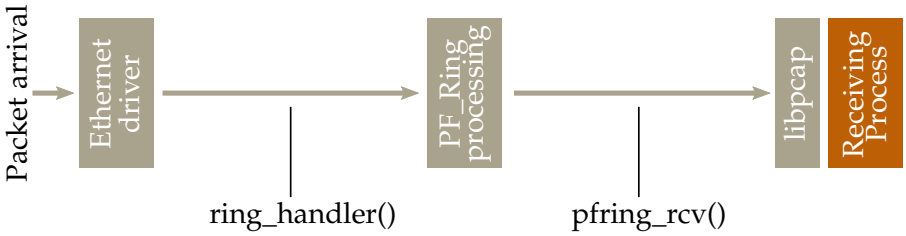


Figure 22: PF_RING processing pipeline on Linux (Braun et al. [72]).

When using a Co-Processor, efficient forwarding of the data to the Co-Processor is imperative [72]. For the Linux kernel, there are several different possible implementations of packet processing. The default packet processing pipeline can be seen in Figure 21 based on the packet interface on device-level *PF_PACKET*. Packets are captured by the hardware — the network interface card (NIC). Then, the network driver forwards the data to the operating system, which hands it off to libpcap, which then sends the data to the capturing application. As can be seen, the pipeline includes several layers of abstraction with different libraries and buffers in between. This design includes interrupt handling and operating system scheduling of kernel threads. This is okay for normal operations but constitutes a considerable bottleneck when working with high-throughput applications. Therefore, PF_RING [109] was invented to improve the packet processing efficiency. Figure 22 shows how this is done. PF_RING streamlines the process considerably by completely circumventing the standard Linux networking stack. It improves performance over PF_PACKET, especially when many, small packets are involved [86, 109]. Both implementations start with the Ethernet driver of the network interface card and end with the libpcap packet capturing library. Therefore, for application developers, both versions can, in principle, be used interchangeably. The TNAPI driver extension for Linux offers additional improvements in terms of memory management between drivers and kernel. While in previous implementations, memory often had to be allocated time and time again for new packets arriving, TNAPI efficiently reuses already allocated memory. TNAPI is designed to be used in conjunction with PF_RING.

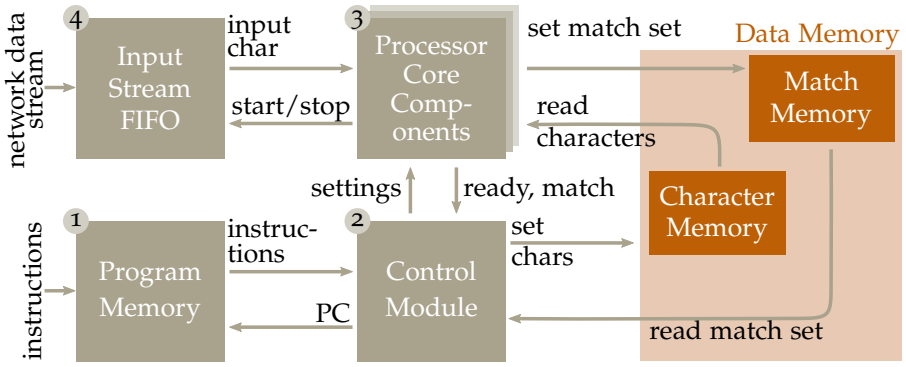


Figure 23: Harvard architecture of one regular expression Co-Processor core. Based on [32].

The packet processing requires the data stream to be copied into RAM by the CPU, then be reread and processed. This is a rather simple task, but it occupies the CPU, which slows down traffic processing. If additional hardware modules such as Co-Processors are used, and the packet processing is not done in the CPU, copying the traffic to RAM might even be unnecessary as these modules often have their own memory. *Zero-Copy* is a model where this superfluous copying is reduced to a minimum by not tasking the CPU to copy the data but using other mechanisms such as shared memory between, for example, the CPU and GPU or direct communication between the NIC and the Co-Processor. Ntop PF_RING ZC (Zero-Copy)[2] is a proprietary implementation of the Zero-Copy mechanism extending PF_RING.

8.2.3 *ASIC RegEx Co-Processor*

Kateryna Shymbarova [32] has contributed to this section with her master project.

The regular expression pre-filter based on FPGAs suffers from the rather low maximum clock frequency of FPGAs (e. g., the Virtex 7 used in the NetFPGA-Sume card has a clock frequency of 200 MHz using our implementation). We therefore also looked into building an application-specific integrated circuit (ASIC) based processors to facilitate the matching mechanisms following the Co-Processor design. The basic design of our concept can be seen in Figure 23. The regular expressions are transformed into a special assembly language understood by the processor. The number of operations for this design is limited compared to general-purpose processors, which leaves a lot more room on the wafer for several hundred cores that can run in parallel. The design of such a processor does not change with changes to the signature database and its ever-changing regular expressions and can, therefore, be realized as an ASIC processor, allowing for lower

2 https://www.ntop.org/products/packet-capture/pf_ring/pf_ring-zc-zero-copy/

Table 8: RegEx assembly code command format.

address	op	type	flag 1	flag 2	operand 1	operand 2
32 bit	3 bit	2 bit	1 bit	1 bit	16 bit	16 bit

production cost in case of mass production and much higher clock frequencies offsetting the lower throughput per clock cycle.

The processor takes the data stream to be matched and the regular expressions in the form of instructions as inputs. The control module (2) starts with the sequential execution of the instructions from the program memory (1). The control module activates the processor core component (3), which is responsible for the instruction and settings — if applicable — are sent out to the component. The processor core component sends the start signal to the input stream FIFO (4) and, in return, receives the input values. The instruction is executed, and the component reports back to the control module if a match was found and that it is ready to receive a new instruction. If the characters do not match, the Co-Processor core discontinues the matching process and reports back that no match was found and that it is ready to receive a new regular expression and new data. Input stream FIFO and program memory are reset. If there was a match, the control module loads the next instruction.

The data memory is split into two parts. For one, the *character memory*. If an instruction has more than two values to match (e. g., (abcd)*), all values will be stored in the character memory, and the processor core component handling the instruction loads the data from that memory block. The *match memory* contains which character matched.

The *control module* controls the process of regular expression matching. If the address bit is set, the addresses may lead to addresses in the program memory only carrying data or to new instructions in the tree structure of the regular expression. In the second case, the address of the instruction currently processed is written to memory on the stack with the instruction to be processed.

A deep tree structure of regular expressions can increase execution time. The parallel execution of several branches is possible with redundant processor core components. However, for the use case discussed in this paper, regular expressions tend to be simple, and parallel branches of a regular expression tend to be rare. Therefore, a parallel architecture on this level would not be beneficial given the additional space requirements.

A basic model for the instruction set can be seen in Table 8. The first bits are reserved for the instruction address as an entry point for possible jump operations. As a second block, the operation indicates the regular expression instruction and tells the processor which submodule has to process the instruction. The operation can be a quantifier such as "$*$" or a logic symbol such as the alternative symbol "|". The

Table 9: Example regular expression assembly code for ((acd)*(b|a))|(c*d).

address	op	type	flag 1	flag 2	operand 1	operand 2
A0	|	A	1	0	A1	A6
A1	*	A	0	0	A2	A3
A2	·	C	0	0	a	c
A3	·	C	0	0	d	0
A4	|	C	0	0	b	a
A5		C	0	1	A8	
A6	*	C	0	0	c	0
A7	·	C	0	0	d	0
A8	<end>					

two final blocks can either contain symbols or addresses, indicated by the *address bit* in the third field. If this bit is set, the final two blocks contain addresses; otherwise, they contain values. If the instruction is short enough that it can be saved in one word, the last two blocks contain the rest of the data. For instance, the regular expression $a|b$ can be expressed in one instruction where the two last blocks contain a and b. If the instruction is too long, the data gets split into several instructions, and the last two blocks contain addresses to the data fields containing the matching strings. This model also allows for nested instructions that can represent the tree structure of regular expressions. Table 9 shows an example assembly code for the regular expression $((acd)*(b|a))|(c*d)$.

Due to the enormous effort to get this mechanism ready for use and the unavailability of resources to implement an ASIC, we decided not to pursue it further. However, we have continued to pursue the basic principle of the Co-Processor, which receives input from the CPU and supports analysis on this basis. Due to far easier implementation and lower cost, we investigated if a GPU could be used instead of an ASIC.

8.3 ACCELERATION WITH GPUS

Mathias Wagner has contributed to this section with his bachelor thesis [38] and his master project [39].

The basic Co-Processor design cannot only be used with ASICs but also with GPUs. GPUs are designed to support the CPU on tasks the CPU is not as capable of performing. The requirements for modern graphics processing are by far exceeding the capabilities of a CPU. Therefore, specialized hardware is needed to tackle the challenges of real-time graphics processing. GPUs are designed to rapidly process and manipulate memory for image creation meant for output on a display device. They are, therefore, capable of highly parallel computation of data stored in matrices. Compared to CPUs, they offer a

lot more processing cores, all the while having a lot less processing power per core. For a long time now, GPUs and their special skills are also being used for applications other than their original intended use under the umbrella term General-Purpose Computing on Graphics Processors (GPGPU). They can be used to mine cryptocurrencies much more efficiently than CPUs and, in that regard, are only lacking behind purpose-built ASICs. They are also often used to train neural networks and to process data in big data applications. GPGPU is even used in high-performance computing clusters [176].

The Compute Unified Device Architecture (CUDA) is an API explicitly published for GPGPU applications on NVidia graphics cards. OpenCL is an open-source alternative supporting a variety of GPUs from different vendors. We decided to use CUDA for our prototype due to the high prevalence of CUDA in GPGPU applications. The GPUs used for this are two Nvidia GeForce GTX 1080 Ti.

8.3.1 *Regular Expression Matching on GPUs*

In principle, both NFAs and DFAs are possible to be processed in GPU cores. However, GPU cores are small; GPUs are optimized for massively parallel computation and not for very complex computations per core. Backtracking, which is necessary for NFAs, would slow down processing, and therefore DFAs are the better choice. The DFA is represented in memory as a two-dimensional array A. The array contains a row for each of the 256 possible characters and a column for each DFA node. If there is a transition from state i to state j with character c, $A[i][c]$ is set to j or $-j$ if this is a transition to a final state. When no transition exists, the array entry is set to the lowest possible integer value *INT_MIN*.

Algorithm 3 Matching packets to DFAs.

```
state = 0;
l = packet length;
while state ⩾ 0 ∧ l > 0 do
    character = next byte in packet;
    state = A[state][character];
    l = l - 1;
end while
if s > 0 ∨ s = INT_MIN then
    return match
else
    return no match
end if
```

The arrays are stored in GPU memory and can then be used to match packets with Algorithm 3. The algorithm has a worst-case

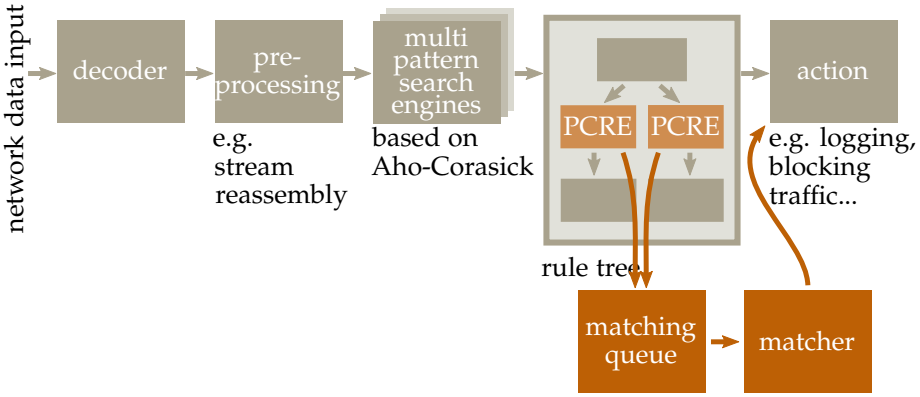


Figure 24: Extended processing pipeline of Snort 3 with GPU matcher module in orange [38].

execution time of $O(n)$ with n being the packet length. One GPU thread is necessary per packet per regular expression. This shows very well how well this process is portable to GPUs. While the algorithm that needs to be executed is straightforward, many of these algorithms need to be executed ideally in parallel to achieve real-time performance.

8.3.2 *Integration with the Snort IDS*

Figure 24 contains the processing pipeline of the Snort IDS and our adaption in orange. Packets arriving at Snort are first decoded. In the pre-processing stage, packets are reassembled into streams to facilitate the analysis of the payload. Snort groups several rules together (e. g., all the rules that are applied to HTTP traffic) and builds Multi-Pattern Search Engines (MPSE) from the search patterns within these rules. These search patterns are based on string matching that is implemented through the Aho-Corasick algorithm. Snort uses these MPSEs in order to filter out any traffic that is guaranteed to not match in the rule tree phase to reduce processing effort significantly. In the rule tree phase, the rules are applied to packets that are not filtered out in the MPSE stage. The most computationally expensive rules that can be applied here are the regular expression matching operations (described in the Perl Compatible Regular Expression (PCRE) syntax, highlighted in orange). The rule tree ends in a decision of match or no match for each rule. Snort can then take action, for example, by logging the event or by blocking any further traffic with the same pattern.

The Snort architecture needs to be adapted to outsource the regular expression matching to the GPU. Matches on the GPU are not executed immediately; instead, they are added to a queue. When the queue is full or a time out is reached, they are sent to the GPU where the matcher runs. For an efficient implementation, the rule tree must

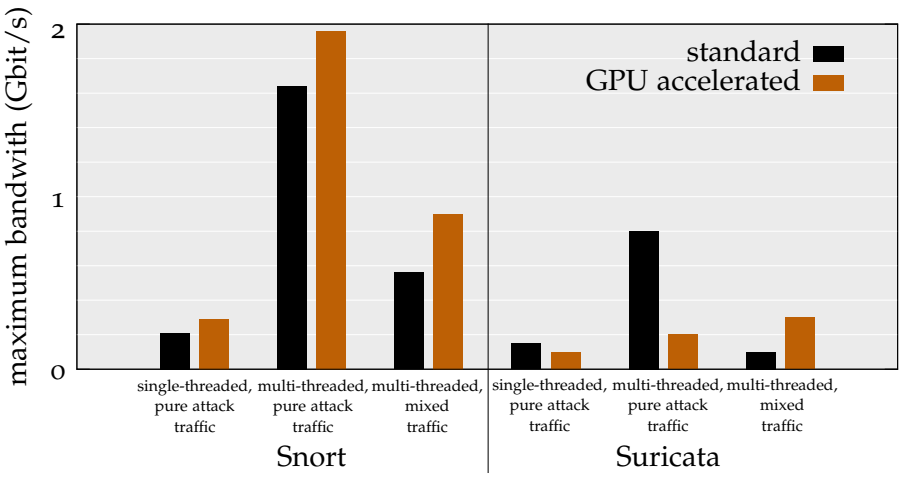


Figure 25: Comparison between Snort 3 and Suricata, single-threaded and multi-threaded with and without GPU acceleration. Based on [38].

not have to wait for the result of the matcher to limit any waiting time and not to let the CPU idle. Therefore, the tree proceeds as if the matcher returned a match. Snort then has to decide whether there indeed was a match during the action phase.

8.3.3 *Evaluation*

We tested our GPU-acceleration implementation for Snort 3 against Snort 3 without GPU-acceleration and Suricata, both with and without GPUs.

The evaluation setup consists of two PCs. One PC running the adapted Snort IDS and one PC replaying the test traffic. The IDS PC features two Intel Xeon CPU E5-2620 v4 in a dual-socket setup having a total of 16 cores, two Nvidia GeForce GTX 1080 Ti, and 32 GB of DDR4 RAM.

Figure 25 shows the comparison of Snort 3 and Suricata. For Suricata, we used the last version that supported GPU acceleration. For Snort 3, we used our own GPU acceleration implementation based on the latest stable version. The attack traffic in this test was tailored to trigger exactly one rule. The mixed traffic consisted of 50% benign traffic produced with iperf3 and 50% attack traffic. We ran both systems with and without GPU acceleration in multi-threaded and single-threaded mode. The graph shows the maximum throughput at which no packet drops could be observed. The graph has no error bars as this measurement was deterministic. Packet drops began at the same throughput in every test run. We can see that Suricata, in this version, both with and without GPU-acceleration, is no match to Snort 3. Suricata shows significant drops in performance with GPU-acceleration when faced with pure attack traffic. Here, the overhead

produced by offloading the calculations to the GPU seems to take up most of the available processing time. Of course, this is no fair comparison as the Suricata version is outdated, while Snort 3 is the newest version that is not even officially released. However, due to these circumstances, going forward, we decided to focus solely on Snort 3.

While insightful concerning Suricata's GPU-acceleration, these test runs were very simple evaluations as we only looked into one rule and only generated background traffic from iperf3. We switched out the background traffic and used the data sets from GPNTF instead as they contain a more realistic scenario and better represent a production network in our extended, more elaborate test runs. In addition, for malicious traffic, the trigger data set was used. This data set is designed to trigger every rule of the Snort community ruleset, showing the effect of vastly different attacks in the network. The traffic is sent by seven instances of tcpreplay. Four of them combined produce 50% of the traffic. The other three share the other 50%. In the beginning, all instances of tcpreplay sent the same benign traffic. After ten seconds, the three instances sharing the second half of the traffic switch to the trigger data set, sending with the same throughput, and switching back to background traffic after the trigger set is done. This is done to assure that the load in the network remains constant and that load peaks in the IDS can be attributed solely to the content of the traffic. Every test was conducted six times. Snort was configured with and without GPU support, with and without Zero-Copy activated, and using the four different available background traffic recordings.

Figure 26 shows the results of the tests that were undertaken. The number of alerts when no packet drops happen, and the system is not overloaded should be roughly 560 000 (as can be seen, for example, at 500 MBit/s throughput for file sharing traffic with Snort without GPU acceleration or Zero-Copy). GPU-assisted Snort highly exceeds these numbers because of false positives. These are due to missing implementations of look-ahead assertions, look-behind assertions, and back-references in the prototype as any evaluation that would need those defaulted to true. However, this had no apparent impact on performance. It can be seen in all tests that GPU-assisted Snort produces higher drop rates earlier than vanilla Snort and misses far more true alerts. Furthermore, activating Zero-Copy, in most cases, did not improve results.

As secondary results, we can observe that using different background traffic has a high impact on the system's performance. While drop rates increase with the activation of Zero-Copy with three out of four different background traffic scenarios, we observed the opposite effect with web traffic. Here, Zero-Copy improved the results. These results underline the importance of using realistic traffic mixes and

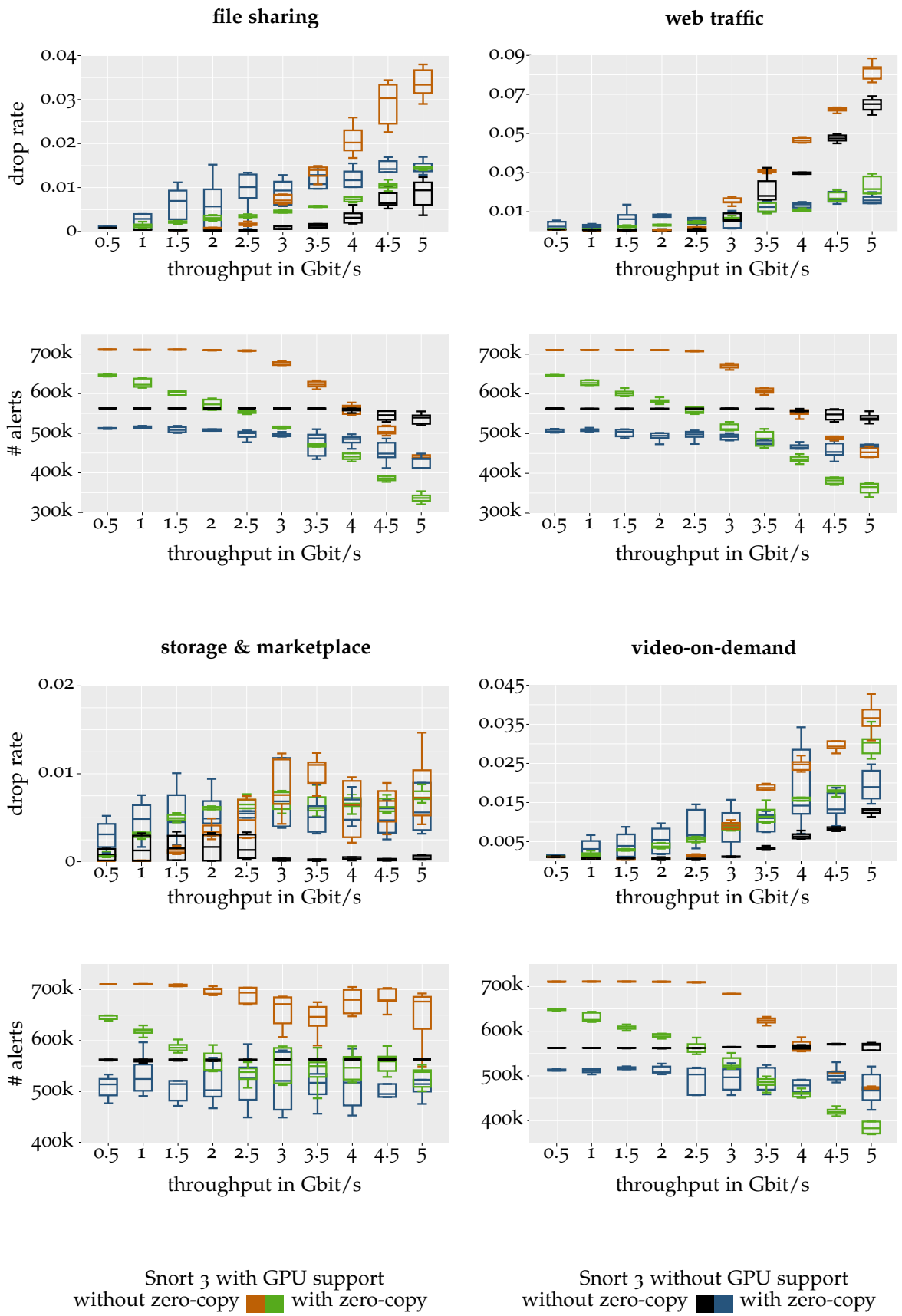


Figure 26: Drop rates and number of alerts dependent on throughput and system. Based on [39].

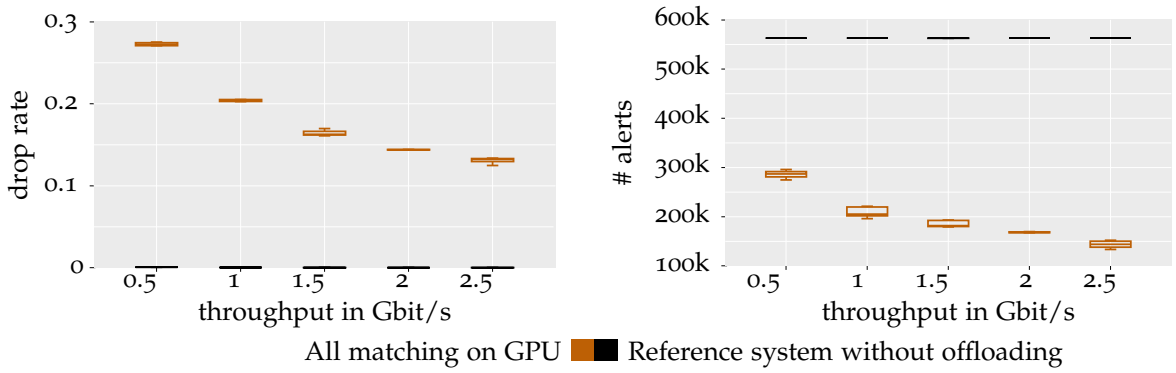


Figure 27: Drop rates and number of alerts dependent on throughput. Background traffic was file sharing traffic; regular expression and pattern matching were offloaded to the GPU. Based on [39].

the dependence of network analysis performance on usage patterns in the network. Storage & marketplace traffic leads to a very low number of dropped packets at any rate. However, this is due to the traffic recording containing many small packets, which leads to the sending server to not reaching the target throughput.

Adding Multi-Pattern Search Engines to GPU Acceleration

Just like regular expression matching, the Multi-Pattern Search Engines can be offloaded to the GPU, too. However, as Figure 27 shows, this decreases the performance of Snort severely. Pattern matching based on the Aho-Corasick algorithm in Snort is optimized to the point where adding any overhead (such as the communication with the GPU) can only decrease its performance.

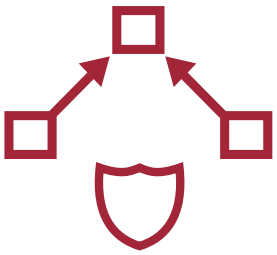
8.4 SUMMARY

In this part, we analyzed Intrusion Detection Systems. We listed their capabilities and their shortcomings. We documented their basic functionality based on value matching on protocol headers, pattern matching for pre-analysis, and regular expression matching. String matching based on the Aho-Corasick algorithm (the Multi-Pattern Search Engines in Snort) is very efficient and runs the smoothest directly on the CPU, further optimization seems unnecessary. Regular expression matching, based on a list of different algorithms beginning with the Thompson's Construction Algorithm and the Powerset Construction Algorithm to convert regular expressions to DFAs, was implemented and optimized to be able to convert it to hardware implementations.

We looked into three different methods of hardware-based IDS acceleration. The pre-filter design featuring the direct conversion of regular expressions to DFAs and then converting these DFAs into

hardware based on the FPGA architecture works in principle and can achieve line speed. However, the limiting factor of the FPGA size reduces the applicability of this solution as the number of rules that need to be supported transcends the number of rules that can be supported immensely. In addition to this, the lack of flexibility and high maintenance requirements limit the operational capability further. Compared to this pre-filter design, the Co-Processor design seems more promising. While the pre-filter requires two separate but cooperating systems (the pre-filter and the main IDS), the Co-Processor reacts to the orders of one master system, which makes it in principle a far more maintainable system that is on top easier to implement. Two different Co-Processor designs were part of our deliberations. As this system does not need to be reprogrammed every time rules change, instead of an FPGA implementation, the Co-Processor can be realized in an ASIC, which allows for much higher clock frequencies. The ASIC-based Co-Processor is a highly complex system where the regular expressions are represented as instruction sets ajar the instruction sets of assembly languages. Based on the NFAs and parallel execution, the regular expression execution can be implemented highly efficiently. However, due to a lack of resources to fully implement such a system, we decided to opt for a more practical approach to acceleration based on GPUs.

The GPU-acceleration of IDS was done already for Snort 2 and Suricata. Although these implementations proofed both the viability of this approach, neither system is currently maintained, and neither the new, upcoming version of Snort — Snort 3 — nor the current version of Suricata features GPU-acceleration. In a first test run, we compared Snort 3 and Suricata, both with and without GPU acceleration. The evaluation analyzed the capabilities of the systems while faced with benign traffic, and a mixture of benign and attack traffic of different kinds. The tests showed that GPU-acceleration could not improve performance in either IDS. As our adaptions not only did not improve the results but worsened them, given our observations, we have to conclude that the usage of GPUs for IDS acceleration cannot be recommended. These observations are in line with the IDS developers offering no GPU support contrary to previous research stating clear benefits. The differences in our results compared to earlier GPU-acceleration efforts such as Gnort based on Snort 2 [307] can be explained by the optimizations Snort experiences over the years. Snort 3 utilizes multiple cores, while Snort 2 ran as a single-core application. Therefore, we conclude that using multiple CPU cores far outweighs the impact of GPU acceleration for IDS.



IV

MITIGATION OF DDOS ATTACKS

9

INTRODUCTION TO DISTRIBUTED DENIAL-OF-SERVICE ATTACKS

Denial-of-Service attacks are a diverse class of attacks with the common goal of targeting the availability of a service. This goal can be achieved by disrupting access to the service or by shutting down the service. One of the basic methods commonly used for these attacks is to bombard the victim with requests, which is especially effective in high-bandwidth networks. The amount of requests then renders the service unavailable to legitimate users. One key aspect to consider for attackers is the efficiency of the attack. For one, the attack has to ensure that the service is unreachable through high resource consumption at the target for maximum impact while the attacker tries to use their resources sparingly. To obtain higher efficiency, attackers usually make use of not only one, but several machines, i. e., the incoming traffic at the target comes from many different sources. These attacks — commonly referred to as *Distributed Denial-of-Service (DDoS) attacks* — are one of the main application areas of botnets. The traffic of tens of thousands of bots can be used to overload a resource on the target. Depending on the specific attack, various resources can be attacked. While *Slow Attacks* are commonly utilized to use up the number of connections a server can open simultaneously, *Reflective Attacks* target the bandwidth of the bottleneck network link. The choice of DDoS attack type depends highly on the specific goal. Some attacks are harder to detect than others but might be less efficient. Others are effective even with a low amount of resources on the attacker side but are easier to mitigate.

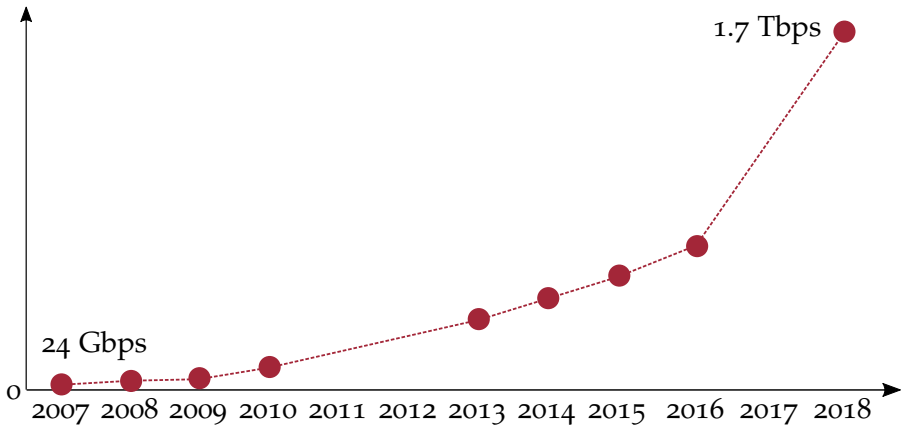


Figure 28: Throughput of the record holder of the biggest DDoS attack of all time [297].

DDoS attacks increased in terms of frequency and throughput over the last decades; they get *"bigger, smarter and more diverse"* [278]. Larger botnets due to the rising size of the Internet, the development of new attack mechanisms, and more available bandwidth are only a few of many contributing factors to this rise of DDoS attacks. Figure 28 shows the record holder in terms of DDoS attack bandwidth for every year from 2007 to 2018 [297]. It can be seen that bandwidth increases rapidly, and stronger attacks can be observed almost yearly. Moreover, with the increasing importance of computer systems for both civilian and military infrastructure, nation-states seem to show more and more interest in conducting DDoS attacks themselves [232, 233], although in most cases, their involvement cannot be proven with certainty. DDoS attacks became so popular that third-party providers of attacks emerged. These services called *stressers* or *booters* attack any target in exchange for money. The vendors usually control a large botnet used for the attacks.

DDoS attacks in general target three different resources of the victim [293]:

- Attacks targeting the *bandwidth* try to flood the network with as much traffic as possible. They try to overload the bottleneck link to the target.
- Attacks targeting *memory* try to exhaust the target's memory capacity by forcing the target to save a massive amount of state information or other data. Attacks targeting a connection limit on the target belong to this category.
- Attacks targeting the *processing power* of the victim are usually application-layer attacks forcing the application, for example, to execute complex calculations (e. g., cryptographic calculations or pathfinding algorithms).

The variety of different attacks with substantially distinct features ensures that general mitigation mechanisms against DDoS attacks are incapable of catching all DDoS attack mechanisms alike. Instead, it is necessary to design different mechanisms tailored to classes of DDoS attacks with similar features.

9.1 BOTNETS

Botnets are formed by a number of Internet-connected devices, usually under common control of command and control software. Access to these machines is most often obtained illegally through security holes [41]. Owners of the machines often do not recognize that the machine has botnet software installed; the devices' normal operations might remain unaffected by the intruders. Especially with the spread

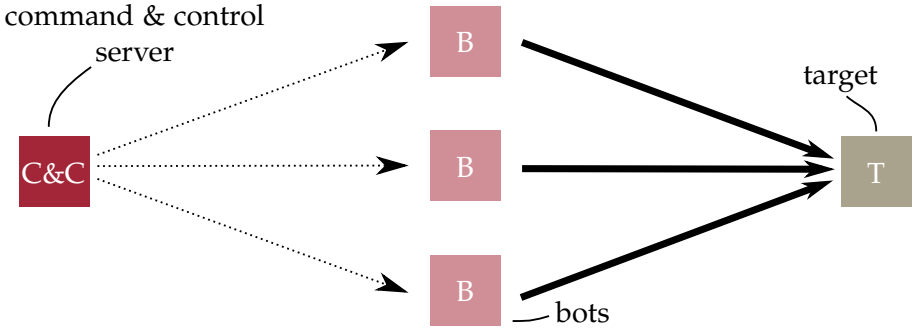


Figure 29: Process of a flooding DDoS attack.

of the Internet of Things (IoT) — i. e., Internet connected physical devices historically not connected to the Internet — and increased connectivity to the Internet of thus far isolated networks including critical infrastructure increased the attack surface significantly. Simultaneously, over the air updates for IoT devices and the application of standard security practices remain insufficiently applied both by vendors and owners of IoT devices alike. The infamous Mirai Botnet reportedly contained 400 000 simultaneously connected IoT devices [179]. The malware tried to infect IoT devices by trying standard login credentials at publicly available ports. The perpetrators were caught, and the botnet was shut down. However, as the source code of Mirai is public, variants of the software are still used to build new botnets. For instance, in an attempt to build a new botnet, 900 000 Deutsche Telekom customers were taken offline as their routers were attacked by a faulty variant of Mirai. Instead of taking over the devices, the software rendered the systems unresponsive [180].

9.2 ATTACK CLASSIFICATION

Literature typically distinguishes between volumetric attacks and application-layer attacks [293]. Volumetric attacks are further divided into whether the attack makes use of a third-party service or technology to amplify their attacks (*amplification attacks*) or not (*flooding attacks*). The term flooding attack can also be found in application-layer attacks (e. g., *HTTP flooding*) besides slow rate or slow-running attacks. In the following, a few examples of such attacks are described.

9.2.1 *Volumetric Flooding Attacks*

The basic principle of a flooding attack, as depicted in Figure 29, is rather simple. A multitude of devices is directed to send as many packets as possible to an intended target to overload a resource at the target. To maximize attack effectiveness, attackers try to achieve

as much resource consumption as possible with as little resource investment as possible. Meaning, the ratio of resources invested in resources occupied at the target should be as small as possible. While the following attacks attack different resources and target different kinds of systems, they all follow this principle.

ICMP FLOODING The Internet Control Message Protocol (ICMP) is a network protocol to distribute metadata in the network. These packets can contain information such as, for example, reachability information of a host or error messages. The control messages can have different types identified in the Type field of the packet header. Type 8, for example, is an *Echo Request* that can be sent to any network node. According to specification, a node receiving such a packet should answer with a packet of type 0 *Echo Reply*. This method is usually used to implement *ping* and can be used to test availability and latency to a target address. In the case of an ICMP flooding attack, attackers send a multitude of Echo Request packets to the victim to generate traffic through the combined bandwidth consumption of the request and response packets. The effectiveness of the attack highly depends on the capabilities of the attacking botnet. As the number of packets can only be doubled, the botnet has to produce at least 50% of the target load itself. An easy way to mitigate the attack is to deactivate *ICMP Echo Replies* on all target nodes and therefore cutting the net traffic of the attack in half. The attack is one of the oldest Denial-of-Service attacks and is very easy to implement. Its effectiveness in relation to its resource demand, however, compared to other attack types, is rather underwhelming.

SYN FLOODING The lower layer SYN flooding attack initially relied on exhausting the limit on half-open TCP connections. The implementation is rather simple: the attacker has to transmit many SYN packets and ignore the corresponding SYN-ACK (i. e., not answering with an ACK packet themselves). The server has to save the state of the connection to be able to open the connection when the expected ACK packet arrives. As servers typically are not meant to keep many of these connections open simultaneously as the three-way handshake usually takes less than a second, in their standard configuration servers often do not have the capacity to hold open many connections at the same time which makes it possible for attackers to exhaust this limit. One semi-standardized protection measure — SYN cookies — requires modification of the host system. Instead of saving the state of the connections on the server, the state is encoded in the sequence number of the SYN-ACK packet. A legitimate client answers with an ACK packet with an incremented sequence number, making it easy for the server to calculate the original sequence number and retrieving the information from the packet. Even with SYN

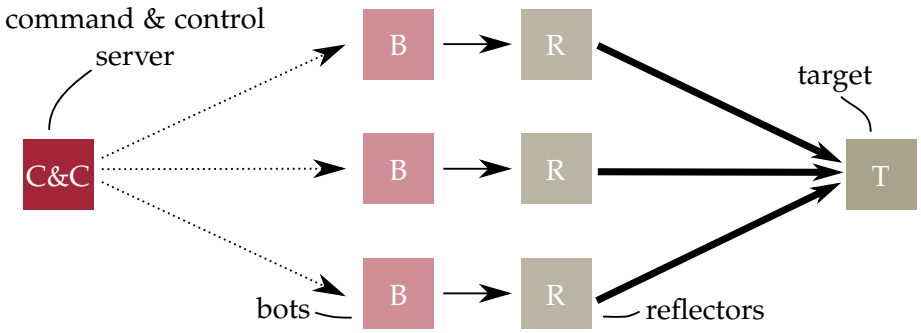


Figure 30: Process of a reflective DDoS attack.

cookies enabled, the attack can still be used to try to overload the bottleneck network link, which makes SYN-flooding attacks still one of the most common DDoS attacks today [168].

9.2.2 *Amplification Attacks*

Amplification attacks exploit third party services on the network to amplify their botnet traffic. These services are neither under control of the perpetrator, nor are they targeted by the attacks. However, their service can still be impacted by the attack as collateral damage.

SMURF ATTACK The Smurf Attack is an advancement of the ICMP flooding attack. Instead of sending the ICMP packets to the victim itself, the packets are sent to a broadcast address. The source IP address is set to the IP address of the intended target of the attack (IP address spoofing). Network nodes now answer the broadcasted *Echo Request* with an *Echo Reply* targeted at the victim's IP address.

The *Fraggle attack* is a variation of the *Smurf attack* where instead of ICMP packets, UDP packets to ports 7 *Echo* or 19 *CHARGEN* are used. In modern network equipment, packets directed to broadcast addresses are no longer forwarded, and hosts should no longer answer broadcast pings. Therefore, this attack is no longer effective in current networks. The basic principle of exploiting third party services on the network to flood a system with their response packets lives on in reflective DDoS attacks.

REFLECTIVE ATTACKS The record holder for the largest DDoS attack to date is a *Memcached attack*. This attack is a Distributed Reflective Denial-of-Service (DRDoS) attack, that does not attack the target directly but instead sends request packets to an exploitable third party service (i. e., the reflector) with a spoofed sender IP address (Figure 30). The third party server's responses are then sent to the actual attack target and cause the overload. Protocols with significantly larger response messages compared to request messages are particularly well suited for these attacks due to amplification effects.

The nature of these attacks requires services that do not need an established connection between client and server. In a recent analysis in 2017, Jonker et al. [162] found that 99.27% of all DRDoS attacks are using the protocols NTP, DNS, CharGen, SSDP, and RIPv1 — all of them are based on UDP. Messages received in a DRDoS attack are hard to differentiate from benign traffic, as they conform to the protocol specification. The reflectors are correctly handling requests they deem to be legitimate. However, the non-existence of a request for the responses is a characteristic of reflective attacks that cannot be masked. Due to the stateless design of UDP, additional mechanisms on the application-layer are often used to attribute a response to a corresponding request. In turn, tracking such request/response mappings within a network is hardly feasible for several reasons, including scalability issues and the necessity to read application-layer messages.

9.2.3 *Application-Layer Attacks*

Application-layer attacks exploit the properties of services. While the aforementioned attacks focus on the network itself and the services necessary for network operations, these attacks target the service directly. For this purpose, the service and its features needs to be known and the attacks in some cases need to be specifically tailored to the target.

HTTP FLOODING HTTP flooding is a classic example of a basic application-layer attack. In an HTTP flooding attack, HTTP requests are sent to a web server. The web server processes the request, prepares a response, and sends it back to the attacker. Sending a request takes comparably little resources, especially if the same request can be sent several times, enabling attackers to invest relatively limited resources themselves. Hereby, the attacker targets the computational power of the victim. The attack can be optimized by requesting a web site at the victim web server that has particularly large resource requirements. For example, pathfinding or navigation algorithms, database access, or services for mathematical function analysis can be lucrative targets.

SSL RENEGOTIATION ATTACK Particularly in early versions of SSL and TLS, a server consumes significantly more resources to respond to a connection request compared to the resources required to send that request[1]. While the usual TLS encrypted communication uses

1 The THC released a proof of concept showing this and related DoS issues in TLS: http://web.archive.org/web/20160311191721/https://www.thc.org/thc-ssl-dos/, exploit can be found here: https://github.com/vanhauser-thc/THC-Archive

symmetric encryption, which imposes minimal overhead both for server and client, the negotiation of the symmetric key uses asymmetric cryptography. The first group making this attack public *The Hacker's Choice* (THC) wrote: *"Establishing a secure SSL connection requires* 15× *more processing power on the server than on the client."* [298]. The client repeatedly requests the renegotiation of the connection, forcing the server to recalculate the keys. One quick fix was to disable SSL-renegotiation; however, this achieves little as the attackers can just open new connections instead. One solution for servers is to limit the number of connections one client can open with the server within a certain time frame.

SLOW-RUNNING DDOS ATTACKS The slow-running attacks are very different from the aforementioned attacks. Instead of overloading a resource by sending many and fast requests, these attacks instead try to slow down the requests as much as possible. Although slow-running DDoS attacks in principle work against other protocols such as IMAP, SMTP, or FTP, the HTTP protocol is the most prominent victim of this attacking scheme. There are three different kinds of slow-running attacks on HTTP:

- The *Slow Header HTTP Attack* is also known as *Slowloris* [283] and is the predominant slow HTTP attack. It was successfully used in 2009 against Iranian government servers [339]. In the Slow Header HTTP attack, a malicious client starts with a regular HTTP request line. After that, the client waits a certain time before it sends an additional custom request header (e. g., *"X-abcd: 1234"*). The client then waits another period and repeats the previous step with another random custom header. According to the specification of HTTP [119], clients are allowed to add such custom headers. This mechanism does not only slow down the initial request. In fact, it does not terminate the request at all. Unless the server applies countermeasures such as a maximum request duration time, an ongoing slow request can bind server resources for an arbitrary period with minimal resource investment on the attacker side.

- The *Slow Body HTTP Attack* is also known as the *Slow POST Attack*, as it relies on the HTTP `POST` method. This method allows the client to submit a request entity such as form data or a file to be uploaded. While regular behavior is used for the request header, the attacker either slows down the transmission of the request entity or provides a *Content-Length*, which is deliberately larger than the actual entity. In turn, this requires the server to wait for additional data. Alternatively, an attacker can use the chunked transfer encoding mode to send arbitrarily slow chunks of a request entity.

- The third variant is the *Slow Read HTTP Attack*. In this attack, the attacker requests a large resource using a regular HTTP request [242]. Once the server starts to send the HTTP response entity, the attacker consumes the incoming stream at a prolonged, low rate, which forces the HTTP server to slow down the transmission due to the small receive buffer [296]. This attack requires much more resources from the attacker as the packets from the server need to be acknowledged and is, therefore, less common than the other two slow-running attacks.

An attacker can run an arbitrary amount of these DoS attacks in parallel both on one computer and on a cluster and can easily exhaust the connection limit of the web server. Due to the low amount of resources and low bandwidth requirements for this attack, detection of the attack within the network can be challenging.

9.3 PREVALENCE OF ATTACKS

Reports on Denial-of-Service attacks show very volatile statistics. Trends change from quarter to quarter. The McAfee Quarterly Threat Reports of 2018 [211–214] list DDoS attacks as the third place of network attacks in March and June, second place in September, and fourth place in their December report. In 2018, The Memcached reflective attack gained traction and broke all records due to its high amplification factor of up to 52 000 [181]. The attack slost track within months when critical design flaws and default configuration errors in the Memcached program were fixed. Other protocols offer a bandwidth amplification factor between 3.8 (BitTorrent) and 556.9 (NTP) [264]. The *Imperva Threat Report* for the fourth quarter of 2017 reported that Internet Service Providers were the most common targets (by numbers of attacks), with gambling sites on the second place [153]. These two targets account for 82% of all attacks. 67.6% of all network-layer attacks lasted less than 30 minutes while the majority of application-layer attacks last between 30 minutes and 6 hours. However, most targets were attacked more than once (67.4% for network-layer attacks, 63.3% for application-layer attacks) with an average of 8.7 attacks and 8 attacks per target, respectively. The most common network-layer attacks are in that order: TCP flooding, UDP flooding, NTP amplification attack, SYN flooding, DNS flooding, DNS amplification attack. The majority of attacks used more than one attack vector. Jonker et al. analyzed four different independent Internet measurement infrastructures over a two-year span [162]. They concluded that at least one-third of active /24 networks had been attacked by a DoS attack in this time frame. Around 70% of all attacks target web services (HTTP and HTTPS). Many attacks target one IP address (37%), around 72% of attacks target at most 10 IP

addresses. 79.4% of the attacks use TCP, 15.9% UDP, and 4.5% ICMP in their data set.

9.4 STATE OF THE ART IN DDOS MITIGATION

Just like the attacks, mitigation mechanisms differ greatly. However, the main steps necessary to be undertaken to mitigate a DDoS attack are mostly the same and can be classified into three different phases.

- *Detection* The first phase is the detection of an attack. This entails not only the realization that an attack is going on but also the identification of the attack mechanism and the identification of the targeted service.
- *Identification* In the identification phase, the network traffic needs to be classified into benign and attack traffic to identify the attacking clients. This step is oftentimes crucial to guaranty that benign clients are not affected by the mitigation mechanism.
- *Defense* In the defense phase, the mitigation system tries to prevent attackers' access to the target service, or at least weaken the impact the clients have on the target. Depending on the viability and precision of the second phase, this step can have a significant impact on the service's usability and availability.

In the following, we will use these terms for these three steps.

9.4.1 *Classification of DDoS Mitigation Mechanisms*

There are many ways to classify mitigation mechanisms against Distributed Denial-of-Service attacks. A common way is to distinguish between the deployment locations of the mechanisms, i. e., where in the network detection, identification, and defense occurs. Zargar et al. [338] differentiate between three different locations in their network:

- Source-based DDoS defense mechanisms are deployed close to the perpetrator. These mechanisms try to find botnets, command and control servers or traffic, or machines in the network that are actively taking part in an ongoing Denial-of-Service attack.
- Network-based DDoS defense mechanisms are deployed anywhere in a network. Neither perpetrators nor victims of DDoS attacks are part of the infrastructure under surveillance. Mechanisms in this category are usually monitoring traffic going

through its network of operation and try to block attack or botnet traffic on its routers. Mitigating DDoS attacks in this scenario is often handled by network operators and offered as a service in the form of DDoS Protection Services that become increasingly popular [163]. A differentiation amid network-based mitigation mechanisms can be made whether the mechanism is transparent for the target host and acts autonomously or whether the target has to request the mitigation from the mitigation service providers actively and has to cooperate for the mitigation to be effective.

- Destination-based DDoS defense mechanisms are located on the victim machines of an attack or in the same local network, specifically defending these machines. For instance, popular mechanisms can change the target machines' settings to make them less vulnerable to the attacks.

- Hybrid or distributed mechanisms are a mixture of two or all three of the above and often require the cooperation of several authorities.

Both for analysis and mitigation, traffic often has to be diverted towards a traffic analysis infrastructure or needs to be removed from the network. Several different technologies can be used for that purpose. Mattijs Jonker [161] mentions two of these technologies in his dissertation. For one, if the protected host is reached through DNS, DNS-based network traffic diversion can be used to redirect the traffic to the analysis infrastructure by adapting the DNS records accordingly. This approach requires the use of a reverse proxy. The second method is the BGP-based network traffic diversion, which is especially useful to protect whole subnets and does not require a reverse proxy. BGP Blackholing can be used to remove traffic from the network destined for a specific target network. This approach is highly effective and can be quickly deployed. However, the downside of this approach is that all traffic is affected, including all benign traffic to the target network. Many of the newer mitigation mechanisms also make use of *software-defined network (SDN)* technology [293]. Software-defined networking refers to the separation of the control plane controlling the network infrastructure and the data plane that forwards the packets in the production network. In general, the idea encompasses a logically central control infrastructure that collects and reacts to data based on a global view of the network. The OpenFlow protocol is one and the most prevalent implementation of the SDN paradigm and is often used synonymously with SDN. However, new protocols such as P4 try to fix some of the issues OpenFlow faces and show that OpenFlow is not the only possible implementation of SDN.

9.4.2 *Source-based DDoS Defense Mechanisms*

One of the most common examples of a mechanism mitigating DDoS attacks is ingress and egress filters based on RFC 2827 [118]. Ingress is the traffic entering a network, egress the traffic leaving a network. Many attacks rely on spoofed source IP addresses to be efficient. Therefore, egress filters that analyze the plausibility of source IP addresses can help limit the effectiveness of DDoS attacks. A source IP address that does not belong to the subnet should be filtered out. Similarly, ingress filters should filter out any source IP addresses that belong to the subnet. Such filters can be easily implemented in any stateless switch or router.

Analyzing the data rate of single machines can also help discover possible machines that are used as bots in an attack [130]. In a small office or home network, the typical upload rate is low compared to the download rate. Therefore, a machine with a high upload rate is highly suspicious. However, it can be tricky to prevent high amounts of false positives in networks with single machines producing high amounts of outgoing traffic such as servers or torrent seeders.

More complicated sanity checks can achieve better results. An IDS can find, for example, frequent occurrences of failed connection attempts typical for worms scanning the network. The system could also use rate-limitation or find suspicious behavior based on the maximum entropy distribution [216].

Systems such as D-WARD [217, 218] constantly monitor the traffic between the subnet and the Internet and compare the traffic fingerprint with stored models of the network. Flows that do not match the usual traffic behavior are rate-limited or entirely blocked depending on their aggressiveness.

9.4.3 *Network-based DDoS Defense Mechanisms*

The network-based defense mechanisms mainly focus on data analysis in the network to detect attacks and to filter the traffic to the victim. With the notable exceptions of some mechanisms such as Mizrak et al. [221] that focus on finding compromised routers in the network, the usual assumption is that neither the perpetrator nor the target of the attack are directly located in this network and in general there is no cooperation of the mitigation system operators with the target.

One approach is to find spoofed IP packets by analyzing the route the packets take through the network [243, 244]. If the route is improbable, an alarm can be raised. Another method is to block clients based on a simple traffic feature analysis [248]. Basically, all hosts are blocked during attacks, that did not communicate with the target recently before the attack.

However, the vast majority of mechanisms try to detect attacks based on commonly analyzed network features. These features include:

- Flow rate [75, 107]
- Bitrate [107]
- Packet Rate [43, 107] (sometimes distinguished between forward and backward packets [91])
- Flow duration [43, 75]
- Entropies based on source IPs, destination IPs, source ports, and destination ports [107, 226, 272, 341] or packet rate [294]
- Ratio of source to destination bytes [43]
- Packet size distribution and mean [91]
- Per packet processing [174, 199]

A broad distinction can be made about how this data is then analyzed. On the one hand, statistical analyses are used to detect attacks [47, 75, 92, 342]. The other major group of mechanisms try to find attacks based on machine learning analysis of the network data [43, 50, 67, 91, 106, 107, 126, 143, 150, 189, 192, 199, 272].

Statistical Approach

Reinforcing Anti-DDoS Actions in Realtime (RADAR) [342] is a system that detects and defends against SYN flooding, UDP flooding, DNS reflective amplification attacks, and link flooding. It detects attacks by analyzing the traffic and finding correlations between attackers. The defense is done by rate-limiting suspicious clients. The evaluation was done with the CAIDA data set in a Mininet virtual setup.

Tao and Yu [294] base their detection method on flow statistics. A low packet rate entropy per flow is regarded as a necessary criterion of an ongoing attack. Additionally, to validate a suspected attack, the information distance of the packet rate between pairs of flows needs to be below a defined threshold. Although the authors validated their detection method by qualitatively comparing it to statistical properties of legitimate traffic and DRDoS attacks, no quantitative evaluation of the detection accuracy exists. Therefore, it is difficult to assess how well data collection and detection will scale to high-bandwidth networks.

Aizuddin et al. [47] propose a system based on sFlow data to mitigate DNS reflective amplification attacks. Their analysis detects attackers by analyzing each DNS request if the same client has already sent a request recently and reacts with rate-limiting of the reflectors.

Zhang et al. [341] evaluate ways of finding suitable thresholds for flow rate entropy detection mechanisms. The evaluation is done in the simulation environment ns-2.

Mousavi and St-Hilaire [226] also look into entropy-based detection of DDoS attacks. However, they focus on attacks on the SDN controller in their network.

FlowTrApp [75] uses flow rates and flow durations to determine whether a client is benign or an attacker.

FlexProject [92] — based on Software-Defined Networking and Network Function Virtualization — focuses solely on SYN flooding attacks. They present a mechanism to prevent IP spoofing. The evaluation is done in a Mininet Setup with Open vSwitch; detection and defense mechanisms are based on iptables and tshark. Benign traffic is emulated with iperf, attack traffic with an unnamed flooding program.

The approach of Wei et al. [322] uses traffic statistics. It correlates the packet rates for all flow pairs passing the same router. Legitimate traffic is assumed not to exhibit any such correlation, while flows belonging to an attack linearly correlate in their packet rate. However, the calculation of pairwise correlation coefficients for each pair of flow makes the detection very costly. According to Wei et al., the quality of the DRDoS detection has a false negative rate of 0.18 %, and a false-positive rate of 0.10 %. Gao et al. [126] conducted their own evaluation of Wei et al. [322], which showed a detection rate of 96 % but a false-positive rate as high as 30 %.

It is noticeable that the analysis of these systems is exclusively done on simulated network environments (mininet if not otherwise stated). The mechanisms have to hold a lot of state information; an evaluation if these mechanisms scale well to bigger systems is missing. A discussion of the scalability of the systems is imperative, especially for attacks with very high data rates, such as reflective attacks.

Giotis et al. [132] use entropy-based anomaly detection on flow statistics from OpenFlow and sFlow data. They conclude that the analysis of full flow data is too extensive and does not scale well. In fact, it could lead the mitigation system to become a potential target of such an attack. Instead, they calculate the source IP entropy, destination IP entropy, source port entropy, and destination port entropy based on sampled data and use these values to detect several different attacks, including DDoS attacks.

Packet Marking and filtering mechanisms try to mitigate DDoS attacks based on historical data. One way is to keep records of which IP addresses showed usual behavior (e. g., complete TCP handshakes) and build a white list of trustworthy clients [186]. This list then inevitably leads to clients who use the service for the first time being perceived as attackers. However, returning users are recognized as

such and can use the service. It is highly dependent on the service and its user group, whether such a restriction is acceptable.

Counting the hops a packet takes until reaching a service can be used as an indicator of whether the supposed IP address can be valid [318]. This, in turn, can be used to detect IP spoofing. However, NATs can falsify this data, and the attacker could spoof IP addresses of similar distance to the target service to counter this detection mechanism.

Some mechanisms require the participation of a large part of the Internet. Path identifier (Pi) [331] attempts to track the paths the packets take through the network by routers marking these packets. The authors state that around half of all Internet routers would have to implement this mechanism for it to unfold its full potential.

Machine Learning Approach

The machine learning approaches commonly found try to use the aforementioned data sources (mostly flow information) and use these metrics to teach a detection system with the use of machine learning. They usually use two different training data sets — one containing benign traffic and one containing DDoS traffic — for the learning phase. Different types of machine learning algorithms are used, from support vector machines to deep learning.

The approach of Braga et al. [67] is based on the analysis of flow features. A six-tuple of statistical features per flow is collected at SDN switches in the target network. Attack features are then distinguished from non-attack traffic that was collected during a training phase. Due to the continuous collection of flow-specific data, in particular from header field inspection, scaling the approach to high-bandwidth networks would require a powerful SDN deployment. The employed machine learning approach *Self Organizing Maps* has a high performance footprint. The author's evaluation showed a detection ratio of about 99 %, and a false-alarm ratio of around 0.5 %.

Gao et al. [126] use traffic features with machine learning to detect attacks. Five defined features are selected from the traffic destined to a target host to train typical network situations. A *Support Vector Machine* detects deviations thereof, which are classified as attacks. The attack detection was evaluated for reflective DDoS attacks and detected over 90 % of attacks with a false-positive rate between 0 % and 8 %. Since the machine learning approach needs to be trained, the detection quality is highly dependent on the training data. This mechanism is resource-intensive compared to other approaches as collecting features from the traffic has a higher performance footprint than using statistical properties of flows.

Li et al. [192] use a deep learning algorithm to detect UDP flooding, SYN flooding, ARP flooding, SMURF attacks, and the pingofdeath. They use the Spirent TestCenter Packet Generator [287] for their anal-

ysis. They report verification rates as high as 98 % to 99 % based on the ISCX data set.

Ahmed et al. [43] propose a mechanism located on the control plane of a software-defined network. They focus their endeavors on tackling DNS amplification and DNS flooding attacks on IoT devices. The approach is very specialized using the number of packets and the connection duration as metrics, and the — very atypical — ratio of source to destination bytes. They base their mitigation method on the *Dirichlet Process Mixture Model*.

FADM [150] uses entropy measurements for feature selection and *Support Vector Machines* for classification. They only consider network-level flooding attacks (UDP, SYN, ICMP). The mechanism is characterized by low response time and quick recovery.

Cui et al. [106] use neural networks to detect UDP flooding, SYN flooding, and ICMP flooding attacks based on outlier detection of the packet rate (incorrectly called packet velocity by the authors).

Alshamrani et al. [50] also focus on the detection and mitigation of DDoS attacks specific to software-defined networks.

Liu et al. [199] analyze traffic based on per-packet processing to counter ICMP flooding attacks.

He et al. [143] compare different classification methods (namely decision tree classifier, random forest classifier, extra trees classifier, support vector machine, and ada boost classifier).

Those and other machine learning based DDoS mitgation mechanisms [91, 107, 189, 272] mostly evaluate their systems based on simulated data, both for attacks and for benign traffic. Sometimes, they do use publicly accessible data sets for benign data. In these cases, data sets provided by the Center for Applied Internet Data Analysis (CAIDA) are often mentioned.

9.4.4 *Destination-based DDoS Defense Mechanisms*

Mitigation mechanisms running on the attack target can easily be deployed by any server operator. Therefore, those with the highest incentive to defend against the attack can perform the mitigation themselves. Countermeasures conducted directly by the server under attack have been receiving a large part of the attention in the literature (e. g., [144, 228, 303]). As server applications terminate connections on the application layer, host-based mechanisms can take advantage of protocol-specific properties and metrics to estimate malicious behavior. For instance, a web server can specify limits for the minimum data rate required for a client when sending an HTTP request. It can also use more aggressive timeout values for the initial HTTP request lines, subsequent header lines, or chunks of HTTP messages.

However, using the machine under attack to mitigate the attack is a double-edged sword. On the one hand, application-based attacks can

be far easier to detect and analyze as the amount of traffic, encryption, or data protection laws might prevent systems within the network to detect or analyze the attack properly. However, there is a risk that using the very machine that is under attack to detect and classify the attack could mean that no mitigation takes place as the machine is not able to act anymore due to the very same attack. Depending on the attack, full mitigation on the target might not even be possible (e. g., some volumetric attacks) because the target of the attack might be the bottleneck link leading to the target and not the target server itself. Under this circumstance, a big part of the attack traffic might not reach the target at all.

Nevertheless, many host-based mechanisms against transport layer DDoS attacks were proposed, for instance, mechanisms based on Management Information Base (MIB) data. MIB data contains routing statistics and other relevant information that can be used to detect attacks on the victim machine and can be used to identify the specific attack [80, 157, 193].

Packet dropping based on the level of congestion is done in *PacketScore* [174], and its extension *ALPi* [55]. Both use packet statistics to compare each packet to benign and attack traffic and rate each packet. Packets that are similar to attack traffic get discarded. The approach requires active network monitoring and, therefore, the data collection impacts the latency of all traffic. The cost for the scoring of packets during detection can be reduced by parallelizing the analysis in a distributed monitoring system to handle the traffic in reasonable time. However, the resource requirements are considerably higher for PacketScore than for other mechanisms, in particular for high-bandwidth networks. The approach was designed to defend against general network attacks and should be applicable to any volumetric attack with a modified set of packet attributes.

IP address spoofing is often used during the execution of DDoS attacks. With falsified source IP addresses, blocking the attackers is harder to achieve as the IP address that is used to identify the attacker can be falsified and changed. Although, according to the Internet Engineering Task Force [56, 118], network operators should circumvent IP spoofing through Ingress filtering within their networks, many ISPs do not implement these measures. Therefore, IP traceback mechanisms try to identify the attackers based on the route of the packets taken in the network [76, 94, 115, 159, 274, 326].

Some mechanisms detect reflective attacks by analyzing the response packets and then decide on the action to take [164, 340]. These methods require the analysis of any response packet until the attack is detected. Due to the high volume of this attack, the defense has to be done within the network infrastructure. However, the target server can provide valuable input.

Many proposals for application-layer attack mitigation try to use the higher amount of data available on the servers for analysis and finding and blocking attacks.

DDoS Shield [255] performs deep packet inspection in a web load balancer preceding the attacked service. This mechanism can work reasonably well for attacks that do not base their effectiveness on their bandwidth consumption but the CPU consumption on the victim service.

DaMask [317] performs mitigation focused on public/private cloud settings. Their DaMask-D module is an anomaly-based attack detection system based on flow information while the DaMask-M module reacts to the attacks. The authors focus more on the principal model of a defense system and provide a framework rather than concrete mechanisms.

Some mechanisms try to perform the defense independently of the attacks and without analyzing the attacks. For example, Sattar et al. [273] use SDN to dynamically allocate additional resources to systems under attack, which requires backup resources to be available while Kampanakis et al. [165] use SDN to implement a moving target defense algorithm, which changes system and network properties (such as IP addresses) to make it difficult for the attacker to sustain the attack. Chen and Chen [93] detect reconnaissance attempts by analyzing SDN flow table entries and mitigates these by providing false information to aggravate the attacker.

9.4.5 *Hybrid or Distributed Mechanisms*

Hybrid mechanisms use a collaborative scheme between the attack victim server or subnet and parts of the broader network infrastructure or network operators. Close integration and cooperation of both groups are often required. The mechanisms aim to profit from the advantages of both. These approaches require modifications to potential victims to implement the reporting, which also presents a potential vulnerability.

One example of such a mechanism is presented by Lim et al. [196], using reports from possible attack victims to block attacks from botnet-based DDoS attacks with standard OpenFlow features. Other mechanisms use information gathered in smaller subnets to change the configuration of routers in upstream networks, implementing aggregate-based congestion control [202, 334] and throttling attack sources as close to the target as possible [95].

L-RAD [177] proposes an active message authentication by deep integration of the target host into the detection mechanism to mitigate reflective attacks. The core idea behind this proposal is that responses to legitimate requests to a UDP service should take the same route back from the server. On the way, these packets are marked. A victim

system can now easily detect attack packets by the markings as the routes do not match.

Defensive Cooperative Overlay Mesh (DefCOM) is an *"example design for a distributed framework for DDoS defense"* [220]. The system implements distributed rate-limiting, handles alert propagation to participating operators, traffic classification, and distribution of resources. Others have also proposed a framework for DDoS mitigation. For instance, *Coordinated Suppression of Simultaneous Attacks* (COSSACK) aims for better connectivity and more information dissemination between network operators [241]. To incorporate multiple defenses in a flexible and scalable way, Mahimkar et al. [203] propose the architecture *dFence* to deploy arbitrary detection and mitigation approaches.

Speak-up is a mechanism that tries to defend attack victims by launching a counter-attack against the botnet [316]. The idea is to increase bandwidth usage of attackers while under attack by sending more data back than usual. The mechanism is based on the assumption that the attacking clients already use their sending capabilities to capacity.

Differentiating DDoS-flooding bots from human users is also a focus in research. A typical example is the use of CAPTCHAs for clients to prove that they are human [44]. This mechanism is limited to be used with a UI and can often be found on the web. Some mechanisms also include a reaction to failed CAPTCHA tests such as blocking suspicious clients [166]. Other mechanisms use, for example, human behavior models that can be used to evaluate the behavior of potential bots [236].

One way to counter application-layer attacks is to rate-limit clients both in terms of the number of connections and in throughput [289]. This approach is reliable when IP spoofing can be ruled out.

Hong et al. [148] have suggested a network-based defense method against Slow HTTP DDoS attacks by using SDNs. Their method introduces an SDN-based defense application that is triggered by a web server but then handles potentially malicious HTTP traffic instead of the web server. The approach relies on assistance by the web server under attack, as the web server actively initiates the attack check routine and forwards message fragments to the defense application and requires access to the application-level payload.

9.4.6 *Denial-of-Service Mitigation as a Service*

The defense against DDoS attacks is the declared goal of many commercial providers of DDoS Protection as a Service applications. An overview of these vendors can be found, for example, at eSecurity Planet[2]. The most prominent players are Akamai, Verisign, Radware, Cloudflare, Arbor Networks, Nexusguard, Dosarrest, f5, Neustar, and

2 https://www.esecurityplanet.com/products/top-ddos-vendors.html#chart

Imperva Incapsula. They nearly exclusively provide their services for cloud-based applications of enterprise or government applications and are usually not available for self-hosted websites. Their biggest strength is their sheer amount of resources that makes it hard for any attacker to overload. Network capacity in the area of several terabits per second can handle even the biggest attacks to date. Imperva Incapsula hinders attackers by not revealing the IP address of its clients' servers. All requests are routed through their proxies. Cloudflare reportedly mitigates one DDoS attack every three minutes. However, they do not publish how their mitigation mechanisms work.

Some non-Commercial Internet services providers also offer DDoS protection, namely the German Research Network (Deutsches Forschungsnetz, DFN) calling their service *DFN-NeMo* [111, 112] and *Warden* by the Czech research network operator CESNET.

There are quite a few tools that can be deployed on the server to mitigate attacks. Probably one of the best known is Fail2ban[3], which is an Intrusion Prevention System also capable of mitigating application-layer flooding DDoS attacks simply by banning clients that open an unusual high amount of connections. The load-balancing proxy HAProxy [184] offers ways to mitigate attacks e. g., by rate-limiting requests. Similar systems are, for example, DDoS Deflate, the Apache mod_evasive module, FastNetMon, DDOSMON, and can also be found as built-in mechanisms in Nginx.

9.5 SUMMARY

Denial-of-Service attacks are as diverse as effective. Mitigation mechanisms have to be adapted to the specific type of attack. While mitigation mechanisms against some of these attacks are very effective, rendering the attack useless in contemporary networks, other attacks such as slow attacks and reflective DDoS attacks continue to be a menace to service providers. While work in this area is extensive, there are several challenges still to be tackled in the area of DDoS mitigation mechanisms. Mitigation often requires the cooperation of the attack target, especially for application-layer attacks. However, especially for DDoS as a Service providers, access to the target might not be possible or prohibited, which means that there is a need for purely network-based mechanisms. Furthermore, defense mechanisms often entail the removal of large parts of the network traffic (e. g., through blackhole routing), including benign traffic. While this is effective against the attacks, it also disrupts normal network operations.

3 https://www.fail2ban.org

10 PROBLEM STATEMENT

Denial-of-Service attacks are still one of the primary threats in network security. Higher data rates, more extensive, more complex networks, and an ever-increasing number of connected devices lead to an increasing significance of attacks. Both size and complexity of attacks increases. While early attacks were made by hobbyists, nowadays nation-states and criminal organizations such as DDoS as a Service providers incentivize the development of new, bigger, and more successful attacks. Mitigation of these attacks is a prolonged endeavor nowhere near its conclusion.

Larger and more sophisticated attacks also lead more and more to small service providers no longer being able to deal with them on their own. More and more services are being offered by network operators and purchased by service operators aiming to defend against DDoS attacks in the network. Network-based mitigation mechanism offered by the Internet Service Providers will increasingly be part of standard business contracts. This means that mitigation mechanisms deployed initially on the target system need to be deployable solely in the network.

The use case that we are considering for our mitigation design is the research network of Baden-Württemberg (BelWü), its decentralized design, and its infrastructure. Several universities, universities of applied sciences, schools, and other public institutions are connected by this service with the Internet. Although BelWü offers hosting of web services, many affiliated institutions — such as research institutes — operate and maintain their own services. The services each institution operates are very diverse, both in terms of underlying operating systems, software products, and operational objectives.

In this setting and with the premise to support BelWü in their endeavor to protect their clients, we try to improve network-based DDoS mitigation. In the literature review, we identified three core mechanisms of successful mitigation. For one, the attack has to be detected. In a second step, the attackers need to be identified, and in a third step, a defense against the attack has to be deployed. Analyzing the state of the art revealed some major points where research can be improved to provide better mitigation services:

- Looking into the mechanisms discussed in Chapter 9 revealed that although many *detection* mechanisms exist and they seem to reliably detect attacks, a contemporary analysis under which circumstances these mechanisms work reliably is missing. The resources available in the research projects bwNET100G+ and

bwNetFlow in combination with other available resources (such as the data sets provided by CAIDA and WIDE) present the ideal opportunity to contribute to this area of research by analyzing some detection mechanisms in the context of research networks. Moreover, most mechanisms in this area work based on passive data measurements, mainly on flow data [43, 47, 50, 67, 75, 91, 92, 106, 107, 126, 143, 150, 189, 192, 272, 342]. Active measurement of potential attack victims is an area that has received little research attention.

- Considering the *identification* of attackers, while flooding attack clients are relatively easy to identify due to their conspicuous behavior, slow attack clients are harder to find. For slow DDoS attacks, the state of the art focuses on cutting long connections or performing deep packet inspection on the target [144, 228, 303] or at least need the cooperation of the attack target [148]. Cutting long connections can lead to false positives as connections from clients with bad Internet service are cut. Deep packet inspection cannot easily be adapted to network-based mitigation as encrypted connections render it unusable.

- The common *defense* mechanisms against Denial-of-Service attacks can often lead to a lot of benign traffic being blocked (e. g., in case of BGP blackhole routing [161]). Especially when we look at the defense of reflective attacks, the defense often means that whole ports are blocked or even UDP as a whole [112]. As a result, the attack victim is unable to use the service that was used for the attack.

10.1 RESEARCH QUESTIONS

Based on these considerations, the following research questions arise:

1. Under which circumstances do common detection mechanisms detect DDoS attacks reliably?

2. Can the common detection mechanisms help to find DDoS attacks in the BelWü network and other research networks with similar characteristics?

3. How can the network-based identification of DDoS attacks be improved? Our focus here lies mainly on slow attacks, as here, the need for improvement has been identified.

4. How can the network-based defense against DDoS attacks be improved? We focus here mainly on reflective attacks.

In order to obtain the necessary results to answer these questions, several steps are necessary:

- Recording and analyzing data in the BelWü network infrastructure. This data can be obtained from the bwNetFlow project.
- A framework that can be used to test the mechanisms meant to improve DDoS mitigation.

11

DDOS MITIGATION FRAMEWORK

Successful mitigation of Denial-of-Service attacks requires an extensive system to detect the attack, identify attackers, and defend against the attack by removing the threat from the network traffic. In the following, we introduce our mitigation system.

The mitigation system consists of three separate groups of mechanisms:

DETECTION Two different detection mechanisms are the focus of this chapter: for one, the detection mechanism by response time measurement of the potential target. This method guarantees that all DDoS attacks that are successfully impairing the target service are found — given that the resource that is checked by the detection system is the one under attack. However, this method is not capable of finding out which attack is running and cannot distinguish between an attack, a flash mob, or other reasons for server downtime. Mechanisms reacting to detection based on this measurement need to keep this in mind when reacting to the attack. The second mechanism is based on flow data analysis and is a common method for finding attacks. Based on flow data and derived metrics such as the entropies of source and destination IPs, or source and destination ports, or the number of flows in the network can be used as indicators of large scale attacks in the network. This method can only be used to detect volumetric attacks as — depending on network size — a large volume of attack flows is required to change the metric.

IDENTIFICATION OF ATTACKERS Two mechanisms to identify attackers are part of the system. For one, a *scoring mechanism* that classifies attackers based on their presumed impact on the target. The fundamental idea behind this mechanism is that attackers tend to inflict more resource consumption on the target than regular clients. This assumption makes sense for resource depletion attacks such as flooding attacks, as the very idea of these attacks is to overload at least one resource on the target. Maximizing the impact for an attacker means that attacking clients exceed the usual resource consumption significantly. This mechanism relies heavily on counting packets and, therefore, on the assumption that DDoS attacks have high packet frequencies. This assumption is not valid for slow DDoS attacks. As slow attacks break with the usual patterns of DDoS attacks — especially in terms of induced traffic amounts — a specialized identification mech-

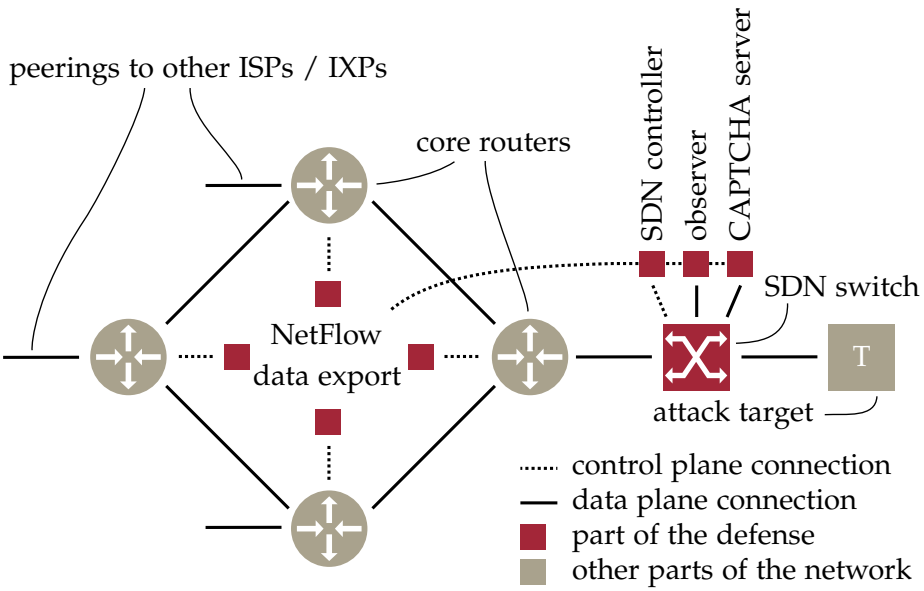


Figure 31: Setup of the full DDoS Mitigation System

anism is necessary. Therefore, the second mechanism is a specialized mechanism focused on slow Denial-of-Service attacks.

DEFENSE The defense is done in two different ways depending on the attack. For one, the *general defense mechanism* is based around the assumption that the attackers could be correctly and reliably identified in the identification phase. This assumption is not feasible to achieve with very high load attacks such as reflective DDoS attacks. For those, analyzing all clients and identifying attackers would be a task far exceeding the computational power of the normal network operations. Therefore, a *specialized mechanism for reflective attacks* that can work without identifying attackers follows subsequently.

First, the environment in which the framework is setup is described. Then, the detection of attacks, the identification of attackers, and the defense mechanisms each for different attacks are introduced. The chapter concludes with an evaluation of all these mechanisms.

11.1 ENVIRONMENT

The system we are looking into is a network-based mitigation system set up within the network infrastructure. Potential targets in the network infrastructure are known by the defenders. However, they are neither in contact with nor controlled by the DDoS mitigation administrators or the mitigation service. Figure 31 shows a simplified, schematic view of the environment in which the mitigation system is set up. In red, the mitigation system itself is shown, while the gray parts represent the parts of the network infrastructure

that are directly connected to the mitigation system. On the left side, the data aggregation based on information from the network's core routers is shown. The Baden-Württemberg Extended Lan (BelWü) — among other parts — contains several core routers connected to other ISPs (e. g., the Swiss research network SWITCH) and Internet Exchange Points (IXPs, e. g., DE-CIX in Frankfurt) as peering partners. The bwNetFlow project is a research project financed by the state of Baden-Württemberg and focuses on the realization of an interface between the core routers to collect flow information, establish an automated processing platform, and detect anomalies[1]. The project exports the NetFlow data of the core routers, aggregates the data, enriches the data with additional information, and provides the data to subscribers. This data can then be used by the DDoS mitigation system. On the right, the mitigation system close to the servers we want to defend — the attack targets T — is shown. SDN capable switches in front of the targets provide the necessary flexibility to realize effective mitigation. An SDN controller controls the switch and can forward attack traffic to the observer for analysis or drop traffic identified as attack traffic. A CAPTCHA server can be used to whitelist legitimate clients during an attack.

11.2 DETECTION MECHANISMS

The goal of the detection of DDoS attacks is to facilitate a fast and thorough reaction. Early detection systems need to react fast, all the while being as accurate as possible. Missed attacks mean that no mitigation will be started or will only be started belated — after administrators find out about the attack through other channels. However, a system detecting too many attacks could also be detrimental. DDoS attacks are quite common; still, they are not common enough that a high false-positive rate is acceptable. A reaction to an attack detection could either mean that alarms are raised, or an automatic system could start to defend actively against the attack. While in the first case, a high false-positive rate would be annoying at best or desensitizing at worst, an automatic response system reacting to a non-existent DDoS attack could have enormous consequences on par with a real attack.

In addition to detecting attacks, additional information can already be collected by the detection mechanism. For example, it can identify what kind of attack we are facing. As DDoS attacks are quite diverse, it is vital to recognize which attack is — or which attacks are — currently running to facilitate the right response.

The two different detection mechanisms that are the focus of this chapter are the detection mechanism by response time measurement of the potential target and the detection mechanism based on network

1 https://www.alwr-bw.de/kooperationen/bwnetflow/

behavior based on entropy measurements. In the following, we will describe how these metrics are set up and how they work.

11.2.1 *Detection Mechanism based on Target Availability*

Alexander Hunt [24] has contributed to this section with his bachelor thesis.

Alexander Hunt, Denis Wagner, and Christian Stehle [25] have contributed to this section with their master project.

Parts of this section have been published at IEEE LCN 2017 [4]

To determine whether an attack is in progress, we determine whether the protected service is still available. Many services in the current Internet are HTTP-based, which provides additional features to detect attacks beyond standard flow-based detection. The HTTP protocol includes status codes that we can use to determine whether a particular service is still available; we use such HTTP requests and measure the response time. From this data, we determine the load status of the webserver. Although this is also theoretically possible with non-HTTP services, HTTP allows us to distinguish between different types of delay. This requires carefully chosen probing URLs; in some cases, the server may only respond to requests for static pages, but fail to generate a dynamic page. Similarly, it may send some TCP segments just in time but is unable to complete the full request in adequate time. Thus, three different types of delays are taken into account based on the recommendations of Mirkovic et al. [219]:

- *whole delay*, t_{whole} is the time between the last HTTP request segment that is sent and the last HTTP response of this request that is received up to the point of measurement.
- It is further divided to measure the *partial delay*, t_{partial}. The first partial delay is the time between the last request segment sent, and the first response segment received. All further partial delays are between two response segments.
- We also take the *TCP round trip time*, t_{TCP} [251] into account. It is necessary because the server may fail to receive or to acknowledge received request segments just in time. Therefore, we measure the delay between sending a packet and receiving the corresponding ACK, including the three-way handshake. Retransmissions do not reset the associated timeout.

These values are continuously measured. If one of the delays exceeds the corresponding timeout, or the server answers with HTTP 503 [119], we consider the transaction as failed. Note that the delays are also applicable to other common protocols, including DNS, Telnet, FTP, and ICMP. The load l_i of the server after transaction number i is calculated with:

$$l_i = (1 - \alpha) \cdot l_{i-1} + \alpha \cdot t_i \tag{13}$$

where $l_0 = 0$, $\alpha \in [0; 1]$ is the weight ratio and t_i indicates if transaction i is successful ($t_i = 0$) or failed ($t_i = 1$). If the load status exceeds a threshold τ_l, we assume the server is under attack.

The weight α represents a trade-off between reaction time and accuracy. One transmission may fail for other reasons than an attack (e. g., congestion in the network), so an immediate reaction ($\alpha = 1$) could lead to a significantly high number of false positives. Therefore, the load l_i should only increase when multiple failed transactions occur in a short time. We have parameterized the timeout and the threshold value so that the speed of the detection can be adjusted appropriately to the network. The observer module — handling any reaction to the attack — is regularly informed about the load status.

11.2.2 *Detection Mechanism based on Network Behavior*

Analyzing the flow data of networks is a common way to detect DDoS attacks. As already mentioned in Chapter 9, several different flow features can be used to detect attacks. In the research network of Baden-Württemberg, we have access to flow data based on the NetFlow standard. NetFlow data is commonly obtainable in networks and, therefore, a reasonable base for detection mechanisms.

Available Data

The NetFlow data exported by all peering routers of the BelWü network are collected at a sampling rate s of $s = \frac{1}{32}$, limited by the capabilities of the routers. The data is then collected and enriched with additional meta-information not inherent to the NetFlow protocol. A full list of available data fields can be seen in Tables 20 and 21 in the Appendix. The fields important for the detection of DDoS attacks are the destination IPs (DstAddr), the source IPs (SrcAddr), the destination ports (DstPort), the source ports (SrcPort), and the protocol (Proto). An identification number (Cid) identifies the subnets where the flow in the network can be associated with (e. g., 10109 is assigned by BelWü to flows to or from the Ulm University network). Only one such identification number is necessary as only outgoing and incoming flows from and to the BelWü network are included in this data source. This limitation means that it is not possible to find attacks that both target a device within the BelWü network and originate from the network.

Entropy Calculation

The Shannon entropy or Information Entropy H is the average rate of information produced by a stochastic data source and can be interpreted as a measurement of information density. Equation 14 shows

how the entropy is calculated with the probability mass function P with each data value $x_c \in X$ with n values x_1 to x_n.

$$H(X) = -\sum_{c=1}^{n} P(x_c) \log_b P(x_c) \tag{14}$$

This information entropy is often measured in bits, especially if the base of the logarithm b is set to 2. The entropy value can then also be interpreted in the way that it gives us a measurement of the number of bits necessary to encode the data set. For example, the entropy of a coin toss would be 1 bit, as 1 bit is enough to encode the two possible outcomes of the toss: heads and tails. If X is not equally distributed, rarer events are containing more information than frequent events. Therefore, a data set containing many rare events has higher entropy.

The data we can extract from our networks can be seen as a sequence of value pairs, one value being the timestamp, the other value the value of the data point. For example, the list of source IP addresses contains value pairs with both the timestamps of the flows and the source IP address. We define such a sequence of value pairs S with the n timestamps t_i and the values s_i:

$$S = ((t_0, s_0), (t_1, s_1), \ldots, (t_n, s_n)) \tag{15}$$

To calculate the entropy of this network metric during a certain time span between the timestamps T_1 and T_2 with $T_2 > T_1$, we take the sequence $\hat{S} \subseteq S$ with

$$\hat{S}(T_1, T_2) = ((t_k, s_k) : T_1 \leqslant t_k < T_2) \tag{16}$$

From this sequence $\hat{S}$, we take the set of unique values $\Sigma = \{s_i\}_{i \in 1,\ldots,k}$ in $\hat{S}$ and define the multiplicity set M with:

$$M(T_1, T_2) = \{m_i : m_i = \text{multiplicity of } s_i \in \Sigma \text{ in } \hat{S}\}$$

ignoring the differences in the timestamps when calculating the multiplicity. From that, we can calculate the entropy $H(S, T_1, T_2)$ with:

$$H(S, T_1, T_2) = -\sum_{i=0}^{n} \frac{m_i}{|\hat{S}|} \cdot \log_2 \frac{m_i}{|\hat{S}|} \tag{17}$$

with n being the number of elements in M. The entropy changes when an attack is in the network. Given for one metric that an attack uses the same value s_a for all its connections, the multiplicity m_a of s_a equals the number of all connections the attack opens. In case of a

DDoS attack, the number of connections $|A|$ can be very high, which can lead to significant changes in the entropy. The entropy under attack changes to:

$$\begin{aligned} H(S, A, T_1, T_2) = &- \left(\sum_{i=0}^{n} \left(\frac{m_i}{|\hat{S}| + |A|} \log_2 \frac{m_i}{|\hat{S}| + |A|} \right) \right) \\ &- \left(\frac{m_a}{|\hat{S}| + |A|} \log_2 \frac{m_a}{|\hat{S}| + |A|} \right) \end{aligned} \tag{18}$$

Where the second part of the term represents the impact of the attack. For simplicity's sake, we measure the number of connections in relation to the number of connections in the benign traffic of the network with $|\hat{S}|$ benign connections and $|A| = a \cdot |\hat{S}|$ attacker connections. This simplifies the calculation to:

$$\begin{aligned} H(S, A, T_1, T_2) &= - \left(\sum_{i=0}^{n} \left(\frac{m_i}{|\hat{S}| \cdot (a+1)} \log_2 \frac{m_i}{|\hat{S}| \cdot (a+1)} \right) \right) \\ &\quad - \left(\frac{m_a}{|\hat{S}| \cdot (a+1)} \log_2 \frac{m_a}{|\hat{S}| \cdot (a+1)} \right) \\ &= - \left(\sum_{i=0}^{n} \left(\frac{m_i}{|\hat{S}| \cdot (a+1)} \log_2 \frac{m_i}{|\hat{S}| \cdot (a+1)} \right) \right) \\ &\quad - \left(\frac{a \cdot |\hat{S}|}{|\hat{S}| \cdot (a+1)} \log_2 \frac{a \cdot |\hat{S}|}{|\hat{S}| \cdot (a+1)} \right) \\ &= - \left(\sum_{i=0}^{n} \left(\frac{m_i}{|\hat{S}| \cdot (a+1)} \log_2 \frac{m_i}{|\hat{S}| \cdot (a+1)} \right) \right) \\ &\quad - \left(\frac{a}{(a+1)} \log_2 \frac{a}{(a+1)} \right) \end{aligned} \tag{19}$$

For this calculation, we assumed that the attack only uses the one value s_a for all connections. However, in real attacks, it also happens that not only one value but a short list of a few values is used (e. g.not only one target IP address but several, but usually not more than ten [163]). Under the assumption that all target values have equal multiplicity and the number of target values c, we can calculate the entropy with:

$$\begin{aligned} H(S, A, c, T_1, T_2) == &- \left(\sum_{i=0}^{n} \left(\frac{m_i}{|\hat{S}| \cdot (a+1)} \log_2 \frac{m_i}{|\hat{S}| \cdot (a+1)} \right) \right) \\ &- c \cdot \left(\frac{a}{c \cdot (a+1)} \log_2 \frac{a}{c \cdot (a+1)} \right) \end{aligned} \tag{20}$$

As already mentioned, DDoS attacks target one or only a few targets at the same time. Therefore, the destination IP entropy goes

down during an attack. It has to be noted that similar observations can be made when a service experiences a sudden rise in popularity (i. e., flash crowd effect), which makes false positives possible, and additional metrics should be taken into account. Network scans can show the opposite behavior concerning the destination IP entropy. While a DDoS attack sends many packets from many sources to only a few destinations, network scans send only one or a few packets to many destinations. Therefore, during a network scan, the destination entropy goes up. There are ways around this that are usually used when performing a network scan. A slow scan is harder to detect. If a very large network is scanned — for example, the entire IPv4 address space — randomizing the order of IP addresses makes it a lot harder to detect the scan as defense mechanisms are only deployed in much smaller subnets, and the entropy in these smaller subnets should not change enough to detect the scan. Therefore, although it is possible to detect some network scans, one should not assume that there are no scans if none could be found by calculating the entropy.

While legitimate clients and flooding attackers send requests to the server, resulting in inbound traffic consisting of requests for the most part, for the particular case of DRDoS attacks, all attack packets are response packets sent by the reflectors. Therefore, during a reflective attack, inbound traffic consists predominantly of responses from the perspective of the victim host. UDP services usually have a predefined port number where they are reachable. Subsequently, responses from this service contain this service port number as their source port. As the attacker cannot control the behavior of the reflectors, this is an inherent feature of reflective attacks that cannot be changed. The entropy of the source port is therefore affected by reflective attacks. Flash crowd effects do not affect the source port entropy, which makes this metric even more valuable.

The entropy of the different metrics in benign traffic is highly dependent on several factors in the network, such as the number of clients, the number of services, the time of day, the amount of traffic, whether there is a holiday, weekend, or a typical workday. This makes it impossible to define clear entropy values for typical non-attack traffic and traffic during an attack. Rather, such values must be set depending on the usual traffic patterns. One way of doing this is to continually calculate the entropies in the network and report a possible threat in case of sudden changes, indicating the beginning of an attack. This opens a possible way of staying undetected by changing the traffic patterns slowly over time. However, threat reports show, that this is highly uncommon in DDoS attacks and sudden starts are the norm [153, 211–214]. Therefore, tracking changes of the entropy should work to detect most DDoS attacks. To analyze changes over time, the timeline has to be split into windows of equal length l. The entropy within a specific time frame is then defined as $H_i(X)$ for time

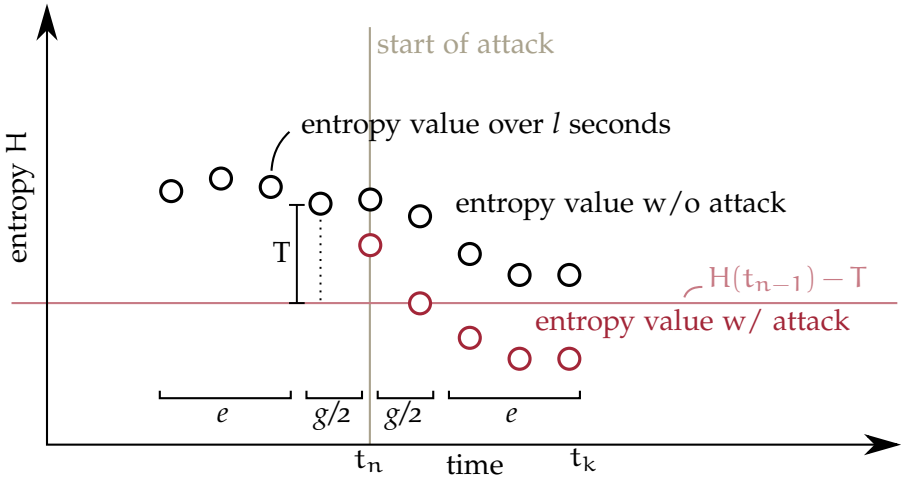


Figure 32: Scheme of the detection approach.

frame i. During an attack, as the traffic is targeted at one specific service, the entropy of the destination IPs decreases. Therefore, it can be observed, that $H_j(X) - H_i(X) \leqslant T$ with $i < j$; i before the attack and j during the attack with a network-specific threshold $T < 0$.

Figure 32 shows an overview of the parameters relevant for the measurements. Entropy measurements are conducted continuously calculated at a point in time t for time slots of length l from $t - l$ to t (black and red circles). At any time t_k, it can be detected if an attack has started at t_n. For this, we calculate the average entropy before t_n as a reference value for a typical entropy observable in this network. For this measurement, two parameters need to be considered: The gap g between entropy measurements and the number of entropy values e taken into consideration. For small l, the transition period between no attack and during the attack might take a few time steps. We take a gap between the measurements g into account to assure that the values that are compared are during the no-attack-time and the attack-times, respectively. This means, we consider measurements that were obtained up to $\frac{g}{2}$ time steps prior to t_n and compare them to values starting at $\frac{g}{2}$ after t_n. First, we take the average of e values at times $t \leqslant t_n - \frac{g}{2}$ into consideration. Taking the average over several values can be beneficial to filter out sudden spikes in the entropy measurements. We opted for the mean of e values instead of median values due to the fact that the median calculation is more computationally expensive. Next, a threshold T is calculated that is below this average entropy value (by subtracting T from the calculated mean entropy). In a third step, the average entropy after $t \geqslant t_n + \frac{g}{2}$ is calculated for e values. If the second entropy falls below T, we consider an attack detected at t_n. We use flow-based entropy over packet-based entropy as large flows within the benign traffic otherwise might distort the results and might lead to more false positives.

If chosen values for the periods l and e are too short, random fluctuations within the network could lead to false positives. Long time frames, on the other hand, lead to a long detection time since a decision if an attack has occurred can only be made after the entropies of the second time frame are calculated. To detect an attack at t_n, the detection time is $t_{det} = (\frac{g}{2} + e) \cdot l + t_{calc}$ with the calculation time t_{calc}. Additionally, as attacks do not start suddenly but also need some time to reach their nominal strength, g has to be chosen accordingly. If g is too large, usual slow changes in the entropies within the network could lead to false positives and an increase in detection time. If g is too small, the entropies that are getting compared could both be within the attack time, and a comparison would not lead to a detection.

This approach scales well because the entropy calculations can be spread out over several network nodes. This works as follows: A load balancer splits the traffic flows to different network nodes. These count the frequency of the source ports and destination IPs and report them to a central instance. The central instance merges the results and calculates the overall entropy. For the source port entropy, a maximum of 65 536 values (maximum number of ports) needs to be considered — independent of the network's actual size.

There are other metrics based on the NetFlow data that can be taken into account. During a UDP-based attack, the ratio of UDP traffic versus TCP traffic R_i in a time frame i increases. A sudden rise of that ratio T_r with $R_j - R_i > T_r$ can be used as a complementary indicator of a UDP-based attack. Based on the commonly known default ports, it is possible to estimate the message type of a request/response-based application-layer protocol (requests use the port number of the service as destination port, responses use the port number of the service as source port). During a reflective attack, the mode of the source port distribution can be used to identify the UDP service exploited in the attack. This information can then be used by a defense system to defend against the attack. Once an attack is detected, the mitigation is activated, and the most used source port (i. e., the port used for the attack) is forwarded to the mitigation system.

To sum it up, the metrics we are looking into based on the NetFlow data are:

- Overall increase of flows to detect network scans and DDoS attacks.
- Destination IP entropy to detect and distinguish between network scans and DDoS attacks.
- Source port entropy to detect reflective DDoS attacks.

- Measuring the dominant UDP source port (i. e., mode of the source port distribution) to identify the attack service of a reflective DDoS attack.

11.3 IDENTIFICATION MECHANISMS

Given that an attack was detected, the defense against the attacks can be initiated. However, many defense mechanisms require knowledge of attacking clients to be effective. For example, rerouting attackers or blocking attackers can only be undertaken if the attackers are known. For this, attacking nodes need to be identified, for example, by their IP address. In the following, two such mechanisms will be presented. For one, a general mechanism against flooding attacks, and a specialized mechanism against slow DDoS attacks.

11.3.1 *General Identification Mechanism*

In case of an attack, observation of the network is enabled for the specific server that is under attack. The SDN controller directs the switch to mirror all packets to the observer where the destination corresponds to this service. The observer can then analyze each client's behavior, which includes information such as the number of open TCP connections. As the observer only analysis metadata of the packets and does not process the requests, especially application-layer DDoS attacks affect the observer far less than the attack target. Nevertheless, enough resources should be allocated to the observer to handle DDoS attacks sufficiently. The load factor of the server, provided by the detection step, is continually updated. For scalability reasons, the observation is only enabled once an attack is detected and only for connections to the server under attack.

During the observation step, every client is assigned a score indicating the degree of suspicion, based on the actions they perform. For each action, the score p_{action} also corresponds to the amount of effort generated for the server by the client — and can be configured individually for every server as needed. Actions that cause more load should be assigned a higher score. The clients are ranked by their score; if an attack is detected based on the load from the detection step, the observer selects the highest-ranked clients as potentially suspicious clients. The score decays after a timeout t_{score}, with a decay factor p_{decay} that represents the sensitivity of the system (i. e., $p_{new} = p_{old} \cdot p_{decay}$). In order to only assess latest network activities, the system observes for t_{score} seconds before taking further action. Otherwise, legitimate clients would carry more weight compared to attackers in the beginning of a slow starting attack.

The potentially suspicious clients are compared to a bound B, visualized in Figure 33, which is initially calculated as $B = p_{max} \cdot \beta$,

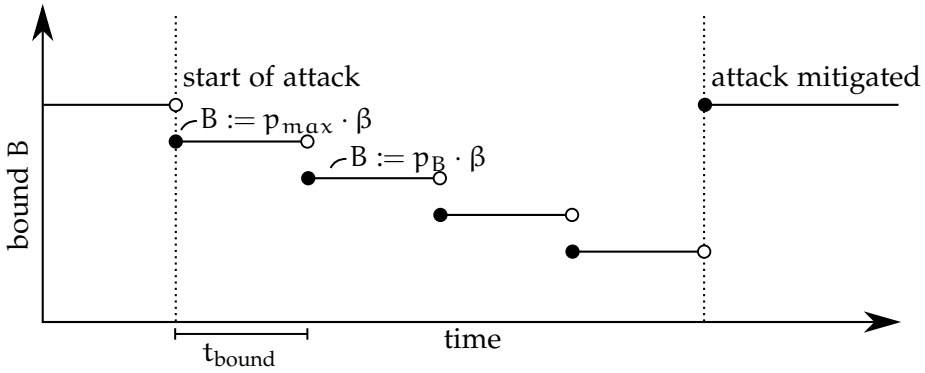


Figure 33: Bound B calculation over time.

where p_{max} is the score of the most suspicious clients and $\beta \in [0;1]$ is a configurable ratio to control the amount of suspicious clients. These clients are then forwarded to the controller and processed. If the server is still overloaded after a configurable timeout t_{bound}, we assume that the attack is still successful and more clients need to be blocked. Therefore, the bound is lowered by computing $B = p_B \cdot \beta$, where p_B is the highest score of clients below the previous value of B. This process continues until no attack is detected anymore. If the bound B reaches 0, all clients are listed as potential attackers. If no more attack is detected, the observation stops, and the bound is reset.

11.3.2 *Special Identification Mechanism for Slow DDoS Attacks*

Parts of this section have been published at EAI SecureComm 2018 [5]

Lisa Maile [27] has contributed to this section with her master project.

Parts of this section have been published at 1st KuVS Fachgespräch Network Softwareization 2017 [6]

Identification of slow attackers is hindered by several factors: The attacking clients behave according to specification, the data rate of the attack is low, and — in case of a highly distributed attack — each client only opens a few connections. This leads to Intrusion Detection Systems not being able to distinguish attacks and regular traffic successfully [313]. Currently, many servers, such as Apache can be configured to mitigate the effect of slow attacks by reducing the maximum time a server waits to receive a full request. However, these changes also block legitimate requests from clients with slow Internet connections, and an attack still has a noticeable impact on the server's performance [228]. Moreover, this mitigation technique requires the administrator to become active; therefore, it is not a viable option for our use case.

Attacks conducted by the most common tools, however, also show common characteristics in terms of network traffic patterns. Based on this traffic, we identified six attacker identification schemes that we considered for our evaluation:

LONG CONNECTIONS (LC) A very basic method measures the duration d of connections and deems very long connections suspicious. This method, however, needs to wait for the connection to last longer

than a certain threshold in the area of minutes and, thus, the identification of attackers can take longer than with other methods. This could also lead to many false positives when the time out is set too short. This is the closest to the already established mitigation method of changing the aforementioned server settings.

LOW PACKET RATE (LPR) One of the core characteristics of slow attackers is a low packet rate p as the attackers try to send as little packets as possible while still keeping the connection open. This scheme alone might also lead to slow regular clients to be blocked. The packet rate is defined as the number of packets divided by the elapsed time since the first packet of the connection.

PACKET DISTANCE UNIFORMITY (PDU) Even time intervals between packets are a feature that can be observed with scripted attacks. The assumption is that non-scripted real clients would send with varying packet rates due to user behavior, network utilization, and available processing resources. Especially clients with bad connections that could be mistaken for attackers by the LPR metric might experience differing packet distances due to non-deterministic packet loss. To reduce load, we only consider the packet distance for three consecutive packets in a row. The packet distance between two subsequent packets is defined as the difference between their receiving time stamps. The PDU is defined as the absolute value of the difference between the packet distances of three consecutive packets abbreviated as Δ.

COMBINATION OF LPR AND PDU (LPR-PDU) LPR can reliably detect slow clients while PDU can reliably detect constantly sending clients while both mechanisms cannot assess the other trait. Therefore, a combination of both schemes could lead to better results. Both traits are present in attackers but should not be common in benign clients. The combination of the two metrics evaluates whether a client shows both characteristics typically attributed to an attacker. As a combination scheme, this scheme requires two thresholds p and Δ.

LOW MEAN PACKET RATE (MPR) The mean packet rate $\bar{p}$ of an attack connection should be high compared to benign clients and could, therefore, also be used as an indicator for an attack if a large number of connections is opened from the same IP.

LOW PACKET RATE VARIANCE (PRV) The mean packet rate of an attack connection should be more consistent compared to benign clients because of the periodically generated traffic for keeping the connection alive. Therefore, we also analyze whether the packet rate variance σ^2 can be used as an indicator for attackers. This is similar

but not the same as PDU as it takes the complete connection time into account.

The chosen schemes require minimal calculation effort, work solely on the network layer with therefore comparatively minor privacy implications, and have very low storage requirements (at max, two values per client need to be stored). In addition to the aforementioned schemes, we also identify two schemes we choose not to evaluate further as they do not fulfill our requirements of a light-weight network-based scheme:

INCOMPLETE APPLICATION-LAYER MESSAGES This scheme operates on the same conceptual level as the approach of Hong et al. [148] for detecting Slow HTTP attacks. Benign clients typically do not send incomplete headers on the application layer as the whole header usually fits into one packet. However, incomplete headers are an inherent feature of slow attacks. Therefore, this method helps to identify attackers reliably. However, it relies heavily on the application layer protocol, a specific detector per protocol and attack type is necessary. For slow HTTP header attacks, for instance, the identification of incomplete packets needs to check *GET* requests for only one end-of-line character at the end or compare the *Content-length* definition with the actual body length of messages. This identification method is quite resource-intensive as deep packet inspection is necessary and requires access to the application layer of the connections. In contrast to the other schemes, encrypted communication (e. g., TLS) cannot be analyzed.

SCORING-BASED MECHANISM A scoring-based system based on our prior work used to mitigate flooding attacks would rate every connection depending on the load caused at the target. For example, a scoring system would rate Slow POST attacks by giving a high score to packets belonging to a *POST* request. The number of connections per single client can be considered to prevent non-distributed DoS attacks. Thereby, any additional connection will increase the score, which is assigned to a client. If other methods fail to identify attackers correctly, this mechanism can at least provide the support for a small subgroup of slow HTTP attacks (non-distributed attacks). Preliminary tests have shown that this scheme is very unreliable for slow attacks and is therefore excluded from the evaluation.

11.4 DEFENSE MECHANISMS

The defense is done in two different ways depending on the attack. The *general defense mechanism* after a successful identification phase and a *specialized mechanism for reflective attacks* that can work without identifying attackers follows subsequently.

11.4.1 *General Defense Mechanism*

After the controller is informed about suspicious clients at the end of each observation step, it directs the switches to redirect all packets with one of the suspicious clients' source IP addresses and the target as the destination to a dedicated server. This server provides individual CAPTCHA tests to distinguish between human users and bots, similar to what Lim et al. [196] suggested. If a suspicious client solves a CAPTCHA, it gets full service access for a duration t_{WL}. On the other hand, if the test is failed multiple times, or too many packets without a solution attempt are sent, they are listed as (confirmed) attackers for a configurable timeout t_{BL}. CAPTCHAs are not always possible to use. In the case of protocols that do not require user interaction and in case of encrypted communication, they cannot be used, and the suspected attackers need to be blocked indefinitely instead.

After identifying which clients are confirmed to be suspicious, there are multiple ways to implement the mitigation of their attack. Traffic-shaping decreases the server load by limiting the data rate, or the number of packets sent to the service. Incorrectly blocked legitimate clients are, therefore, still able to use the service with reduced bandwidth. However, for some attacks, this is not sufficient, as attacks are not necessarily dependent on high bandwidth. For example, some application-level attacks, such as an HTTP GET flood on a dynamically generated page, can overload a server with comparatively little bandwidth. To effectively mitigate such attacks, either the rate limit needs to be very low, or the traffic must be blocked entirely. If Software-Defined Networking capable switches are used, OpenFlow 1.3 [249] and up provides both blocking and rate limitation of specific flows based on data or packet rate. Therefore, we use the switches to perform all mitigation measures. However, note that rate-limiting features are not a mandatory part of OpenFlow 1.3, so some vendors may not implement it, which leaves blocking as the only remaining option.

Our system has multiple mitigation measurement levels. When an attack is detected, the first level with the highest data or packet rate limit is applied to detected attackers. If the server is still overloaded after a configurable timeout t_{level}, the next level with a lower limit is applied. On the last level, the switch blocks all packets from the attackers.

The CAPTCHA server could constitute a bottleneck and become a target of a DDoS attack itself, but it has the advantage that it is entirely controlled by the service provider. This allows the provider to enable protection techniques in a centralized way, instead of deploying these measures for each protected service.

One limitation of mitigation techniques within the network is that they typically rely on IPs to identify attackers. Since these addresses

are not authenticated, an attacker may spoof these for certain types of attacks (e. g., the SYN flood), where the response to an initial message is not relevant to the attacker. IP spoofing is often prevented within ISP networks in today's (mostly IPv4) Internet, limiting the potential of such an attack. However, it is still important to remark that our mitigation strategy only works as presented, if the number of rules to be processed by the switches remains limited. IP spoofing, especially when combined with a SYN flood attack, potentially breaks this limit. In order to deal with this, we have theorized a more aggressive mitigation mode for our framework, where all traffic is moved through the CAPTCHA service, which then performs whitelisting.

Although such attacks may seem theoretical, the wide-spread introduction of IPv6 and the corresponding privacy extensions [231] have the potential to create precisely this issue even if IP spoofing is prevented in the network. Standard IPv6 addresses are generated by a prefix combined with the MAC address of a device; because this would allow global tracking of devices, privacy extensions allow clients to select a random value instead of the MAC address. This allows an attacker to potentially generate many different addresses that are all in the same IP range, allowing an attack similar to IP spoofing, except that the attacker can also receive the targets' responses. A potential solution to deal with this type of attack is to blacklist IPv6 address ranges instead of individual clients, and sporadically whitelist individual clients within that range on the successful resolution of a CAPTCHA. This works, because legitimate clients will use precisely one IPv6 address for the duration of their session, while attackers will use many.

This defense mechanism does not scale well with high traffic rates, which can be observed, for example, with reflective attacks. Therefore, a specialized defense mechanism for reflective attacks is necessary.

11.4.2 *Special Defense Mechanism for Reflective DDoS Attacks*

Parts of this section have been published at IEEE LCN 2018 [7]

Kevin Stölzle [35] has contributed to this section with his master thesis.

Reflective attacks consume the highest amount of bandwidth. Analyzing every connection as we do in our identification mechanisms or blocking individual clients based on their score in a suspiciousness ranking is not feasible with the vast amount of data that needs to be analyzed. Furthermore, the traffic received by the target does not come from attackers but reflectors. As the reflectors are services available to everyone on the Internet and typical protocols used for the attacks are probably also in use by the target (e. g., DNS or NTP), blocking them would also potentially prevent the target from using this service.

Therefore, the focus here has to lie on the flows. While the same IP address can, in one case, be an exploited reflector and in another

providing a service to the target, the flows are clearly either benign or malicious. Identifying malicious flows, however, is hard to obtain. Reflective attacks usually use UDP; therefore, there is no connection state. However, what we do know about the attack packets is that while they are addressed to the target, the target never sent out a request for these responses. Therefore, we need a way to distinguish between response packets sent to the target as requested by the target or by the attacker. As the amount of traffic far exceeds the amount of traffic in usual network operations, analyzing the payload is out of the question, and we have to rely solely on the connection data. One way of keeping track of benign responses and attacker responses would be to track outgoing requests of the target, save the state, and let the corresponding responses through. However, to be as effective as possible, the defense needs to be distributed as far as possible.

As the attack targets the bottleneck link to the target, the attack traffic should be dropped everywhere in the network. Distributing the knowledge of benign requests across the network would be hard to obtain. An easier way would be if the request packets carry the information that they are benign in them. This information needs to be impossible to obtain by the attacker or at least easy to change by the defender if the attacker finds out. Additionally, it needs to be contained in the data that is usually used for packet routing — the 5-tuple of the TCP/IP protocols: the protocol (UDP/TCP), the source IP address and source port, and the destination IP address and destination port. The protocol is set (UDP), the destination IP and destination port are set (IP of the UDP service provider) and cannot be changed to assure that the packet reaches the target, the source port could be adjusted, but collisions are possible and in control of the attackers (different attackers using the same port) leaving the source IP address. The basic idea is that we can tag the outgoing requests from the target to the service by exchanging the target IP address IP_{target} with an alias IP address IP_{alias}. We can limit this only to flows for the service used in the attack to minimize interference in the network traffic. We can block all traffic using the same service as the attack going to IP_{target} as any benign request would use IP_{alias} instead. We can identify traffic using the same service as the attack by identifying the protocol as UDP and the corresponding port of the service that is used in the attack port P_{attack}).

The effectiveness of this approach depends on the distinctiveness of responses and requests. One possibility to differentiate responses from requests would be to analyze the payload, possibly containing a response flag or similar static substrings. Using payload analysis does not scale well and requires protocol-specific knowledge. However, most services that are used for reflective attacks have a default port (e. g., 53 for DNS) that can be used instead [162] as the request to a service uses the service port number as destination port address

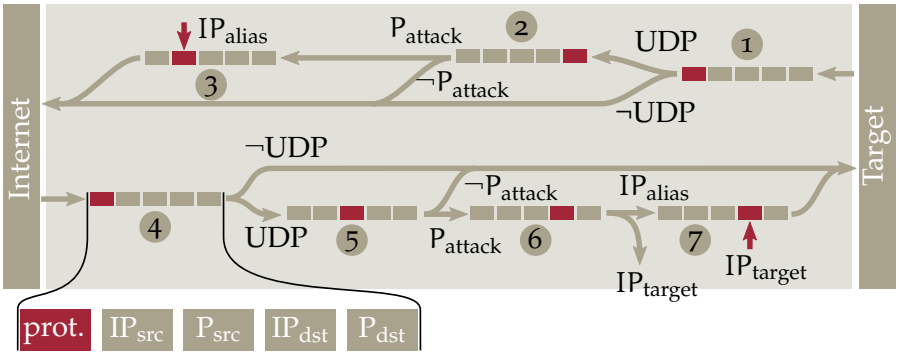


Figure 34: Process of the defense against reflective DDoS attacks on the switch closest to the target. In red: the fields under observation / changed in each step.

while responses use the service port number as source port address. In the rare event that both port numbers are set to the service port number, we cannot identify the direction, and the connection has to be cut. Our detection mechanism is able to extract and identify the application protocol port from the UDP header and to forward it to the mitigation system. This approach is protocol-agnostic and supports any UDP-based request/response protocol that follows a client/server design.

Algorithm

Figure 34 shows how this works in the switch closest to the attack target. We assume all traffic from and to the target goes through this switch; otherwise, the same deployment has to be made on other switches assuring that the full traffic from and to the target gets the same treatment. We only take a look at the traffic from and to the target; other traffic going to other clients is not affected by this mechanism.

On the top, the traffic is shown going from the target to the Internet, while the bottom half shows the way back. (1) Outgoing traffic to the Internet is first analyzed based on the protocol in use (UDP, TCP, etc.). If the protocol is not UDP, the packet is sent to the Internet without further analysis. (2) If the protocol is UDP, we take a look at the destination port P_{dst} next. (3) If P_{dst} equals P_{attack}, in the next step we set the source IP address IP_{src} to IP_{alias}.

On the way back, we again take a look at the protocol. (4) Non-UDP packets are sent to the target without further analysis (given that the reflective attack is the only attack running; otherwise, this traffic needs to be analyzed by the aforementioned general defense mechanism). (5) For UDP packets, we look at the source port P_{src}. If P_{src} equals P_{attack}, we take a look at IP_{dst}. (6) If IP_{dst} equals IP_{target}, we can now safely assume that the packet is an attack packet and

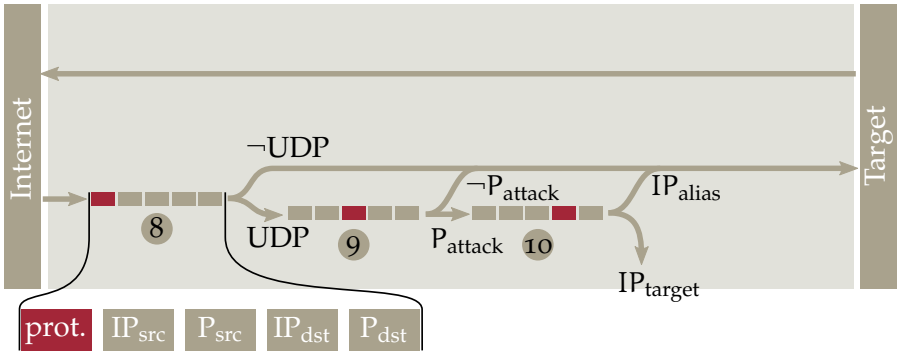


Figure 35: Process of the defense against reflective DDoS attacks on the switch anywhere else in the network. In red: the fields under observation / changed in each step.

can discard it[2]. (7) If IP_{dst} equals IP_{alias}, we can assume that this response is legitimate, set IP_{dst} to IP_{target} again and send the packet to the target.

Figure 35 shows the system how it needs to deployed network-wide or even upstream to other network providers. Here, the packets from the target to the Internet are already tagged, outgoing traffic does not need to be interfered with. (8) On the way back from the Internet, we again look at the protocol and P_{src}. (9) If the protocol is UDP and P_{src} equals P_{attack}, we look at IP_{dst}. (10) If IP_{dst} equals IP_{target}, the packet is dropped. Packets addressed to IP_{alias} are forwarded towards the target. Therefore, the DRDoS actual defense is a simple forwarding unit that discards all UDP packets addressed to the target IP address.

In principle, this mechanism works similarly to a Network Address Translation (NAT) [53, 288]. However, as we only switch out two IP addresses, we do not need to save any state anywhere in the network. This solution scales very well for very high attack packet rates as the attack traffic does not have to be observed, and no state needs to be saved.

It is imperative that IP_{alias} is not easily guessable, so it is not easy for an attacker to switch their attack to the alias IP address. As the alias IP address is only visible to UDP-based services utilized by the target service (and these services, in general, cannot know, that this request comes from that service) the alias IP address can only be found out by the attacker if the attacker can observe the network traffic between the target and the service[3]. Should IP_{alias} be found out by the attacker, only the configuration at the closest switch to the target needs to be changed to use another IP address.

2 With the exception of responses to requests made by the target before the defense system was active. These packets will be falsely discarded.

3 Mind that the attack target uses IP_{alias} only for requests, a potential DNS entry for the target remains unaffected.

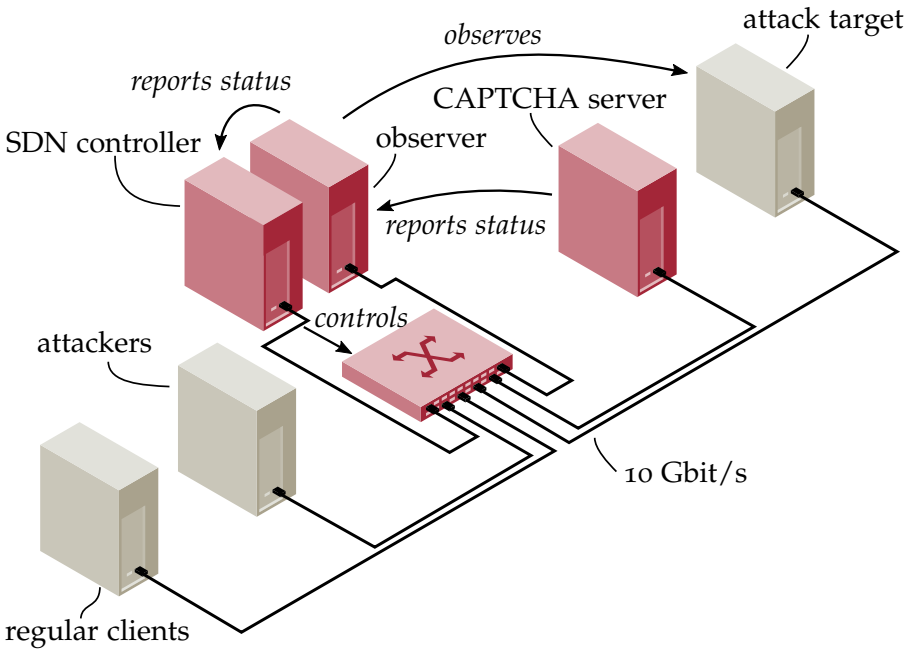


Figure 36: The local setup of the DDoS mitigation framework.

11.5 PROTOTYPE SETUP

For testing purposes, a local test setup was implemented, as shown in Figure 36. The system consists of the local setup of the mitigation system without the NetFlow data export, which is evaluated separately. Additionally, the system entails a web server functioning as an attack target in test runs, one machine simulating attacks, and one machine simulating regular clients. We choose a Software-Defined Networking (SDN) enabled network as our prototype implementation environment as the flexibility of SDN enables fast prototyping independent of hardware vendors. Our system's primary requirement is the ease of deployment in a given network with low overhead and without changing the network topology. TTwo components are needed to implement this system: a monitoring system that can handle the traffic load, and an SDN controller that works with our hardware, allows for rapid prototyping and works well with our network monitor.

Our hardware features an HPE FlexFabric 5920 switch, which supports OpenFlow 1.3. The web server (attack target) and the monitoring system (Ryu and Zeek) run on separate systems, each with the following specifications: CPU with 4x3.10 Ghz[4] and 17 GiB of memory. The attacks are run from three replay server with a CPU with 6x2.40 GHz[5] and 125 GiB of memory each. With this hardware, a total attack throughput of 6.5 Gbit/s could be achieved.

4 Intel® Xeon® Processor E3-1220 v3
5 Intel® Xeon® Processor E5-2630 v3

11.5.1 *Observer*

The observer is implemented based on the Zeek Network Monitor[6]. Zeek was chosen as it already provides an event-based system that was easy to extend. However, the Zeek Network Monitor does not provide active components. Our concept includes active probing of the possible DDoS attack targets. Therefore, we needed to extend its functionality. With our extensions — an additional python script that communicates with the observer — the system is now capable of evaluating whether a server is under attack. After determining the extent and likelihood of an attack, the inbound traffic to the attack victim is evaluated, and the suspiciousness score for each client is calculated. This list is then forwarded to the SDN controller.

11.5.2 *Controller*

For the implementation of the SDN controller, Ryu is used[7], a Software-Defined Networking framework. Ryu presents itself as a good choice as Zeek already offers an interface connection to Ryu (Broccoli; Bro Client Communications Library), and our concept does not place any other specific requirements on the controller. The controller takes the list of suspicious clients from the observer and decides which clients should be forwarded to the CAPTCHA server, based on their suspiciousness score and the severity of the attack. This decision is then forwarded to all switches under its control. If the CAPTCHA test was successful, the client is unblocked and can access the service again. For our evaluation, both the simulated attackers and the simulated benign clients are bots, which means a real CAPTCHA server cannot distinguish between them. Instead, for testing purposes, the CAPTCHA implementation knows which clients are benign and which are malicious, and uses this to simulate a CAPTCHA entry delay. Attackers will be treated as if they could not solve the CAPTCHA. Benign clients will be cleared after 10 seconds, as research by Brusztein et al. showed that this is approximately the average time a real user would take to solve the CAPTCHA [77]. This average time fluctuates and is dependent on the CAPTCHA implementation but does not have a large impact on the overall mitigation effectiveness.

11.5.3 *Reflective Attacks Defense Mechanism*

The defense mechanism against reflective attacks differs greatly from the other defense mechanism. The traffic is analyzed solely within the

6 https://www.zeek.org/
7 https://osrg.github.io/ryu/

SDN infrastructure and, depending on implementation, even solely on the switches.

Dependent on the capabilities of the SDN switches, two different ways of implementing the defense mechanism against reflective attacks are possible in SDN. One solution utilizes the SDN switch to forward packets based on their header fields to the Ryu controller to actively modify packets before they are forwarded. The SDN rules for this implementation can be found in Table 10. This implementation relies on mandatory OpenFlow 1.3 features and, therefore, works with any switch that implements OpenFlow 1.3 correctly. The second implementation makes use of optional OpenFlow 1.3 features, namely switching out the IP address before forwarding and changing ARP fields within the switch without contacting the controller. The utilized rules are shown in Table 11. As this variant works without the involvement of the SDN controller in modifying packets, it is faster and more scalable. However, not all switches implementing OpenFlow 1.3 have the capability to run this implementation.

The red row in both tables represents the distributed part of the defense mechanism. This rule needs to be forwarded in the network and ideally beyond the borders of the network to assure successful mitigation of the attack.

11.6 EVALUATION

To summarize, we built a framework for DDoS mitigation. We implemented detection schemes for DDoS attacks, designed and implemented attacker identification schemes for DDoS attackers, and designed and implemented DDoS mitigation schemes and integrated them into the framework.

In the following, the *overall performance* regarding throughput and reaction times of the framework, including the DDoS detection based on target availability, the identification mechanisms, and general defense scheme, are evaluated based on our local test framework.

Additionally, the *accuracy* of the detection mechanism based on background traffic is evaluated. Based on our analysis in Part II, we chose the data sets CAIDA2019, WIDE Day in the Life of the Internet (DITL) 2019, and our own bwNetFlow data recordings, each with added simulated attack traffic to conduct our analysis. The data sets are both up-to-date and extensive.

Furthermore, the *accuracy* of the identification mechanism of slow attacks is evaluated based on our own data set SUEE1/SUEE8, including simulated attack runs (documentation of this data set can be found in Chapter 5).

Table 10: Defense mechanism against DRDoS attacks without optional OpenFlow features. IP_t: IP address of the target host. IP_a: alias IP address. p_a: default port of the attack.

EtherType	Match Fields IPv4 **SRC**	IPv4 **DST**	UDP **SRC**	UDP **DST**	**Action**
0x0800 (IPv4)	IP_t	*	*	p_a	⇒ CONTROLLER
0x0800	IP_t	*	p_a	*	⇒ TARGET
0x0800	*	IP_t	*	p_a	⇒ TARGET
0x0800	*	IP_t	p_a	*	DROP
0x0800	*	IP_a	p_a	*	⇒ CONTROLLER
0x0806 (ARP)	*	*	*	*	⇒ CONTROLLER

Table 11: Defense mechanism against DRDoS attacks with optional OpenFlow features. IP_t: IP address of the target. $b(IP_v)$: Broadcast address of the target host's subnetwork. IP_a: alias IP address. MAC_v: target MAC address. p_a: attack port.

EtherType	Match Fields ARP **OP**	ARP **TPA**	IPv4 **SRC**	IPv4 **DST**	UDP **SRC**	UDP **DST**	**Action**
0x0800 (IPv4)	*	*	IP_t	*	*	p_a	set-field IPV4_SRC = IP_a ⇒ TARGET
0x0800	*	*	IP_t	*	p_a	*	⇒ TARGET
0x0800	*	*	*	IP_t	*	p_a	⇒ TARGET
0x08000	*	*	*	IP_t	p_a	*	DROP
0x0800	*	*	*	IP_a	p_a	*	set-field IPV4_DST = IP_t ⇒ TARGET
0x0806 (ARP)	1	IP_a	*	*	*	*	set-field ETH_SRC = MAC_v set-field ETH_DST = ff:ff:ff:ff:ff:ff set-field ARP_OP = 2 set-field ARP_SPA = IP_a set-field ARP_SHA = MAC_v set-field ARP_TPA = $b(IP_v)$ set-field ARP_THA = ff:ff:ff:ff:ff:ff ⇒ TARGET

11.6.1 *Performance Evaluation*

Parts of this section have been published at IEEE LCN 2017 [4]

Christian Stehle [34] has contributed to this section with his master thesis.

Several experiments were completed to analyze the performance in our aforementioned test setup and to show the viability of the concept in high-bandwidth networks. The setup contains:

- Detection based on target availability
- General identification scheme
- Both defense mechanisms (general defense and special defense mechanism for reflective attacks)

The following mechanisms are evaluated separately:

- Detection mechanism based on network behavior
- Slow attacks identification scheme

We varied the percentage of attackers for each attack scenario. Overall, 400 clients access the server in every test run, of which 30 %, 60 %, or 90 % are attackers; the remaining clients are legitimate. We repeated each test 25 times to rule out any random effects in the network. In each test, the legitimate clients continuously try to access the server, while the attacking clients are only active when the attack starts.

We identified four points in time that correspond to important events in the system. The first is the start of the attack, t_A, as a baseline for the other events. Then, t_D is the point in time where the mitigation system detects the attack, which also marks the point in time when the analysis of the traffic starts. No recordings or analyses are done before t_D, except for the detection mechanism that finds ongoing attacks. Mitigation completion is defined by the moment the forwarding rule for the last attacker is installed; this time is t_M. The moment when the server has recovered and is available to all clients is defined as t_R. For flooding attacks, if the server is available again before the last attacker is rerouted to the CAPTCHA server, t_M would be infinite, because the mitigation system stops mitigating the attack (and would thus never block the last attacker). We instead define $t_M = t_R$ in this case, because the mitigation essentially ends whenever the server has recovered.

Three time frames are of special importance for an evaluation. For one, the detection time of an attack defined as $\Delta t_D = t_D - t_A$. Furthermore, the mitigation time defined as $\Delta t_M = t_M - t_D$ gives information about the quality of the mitigation algorithm and the server downtime defined as $\Delta t_S = t_R - t_D$ is especially important for the application of this system. Note that Δt_S is dependent not only on the mitigation system but also on the server implementation itself, as

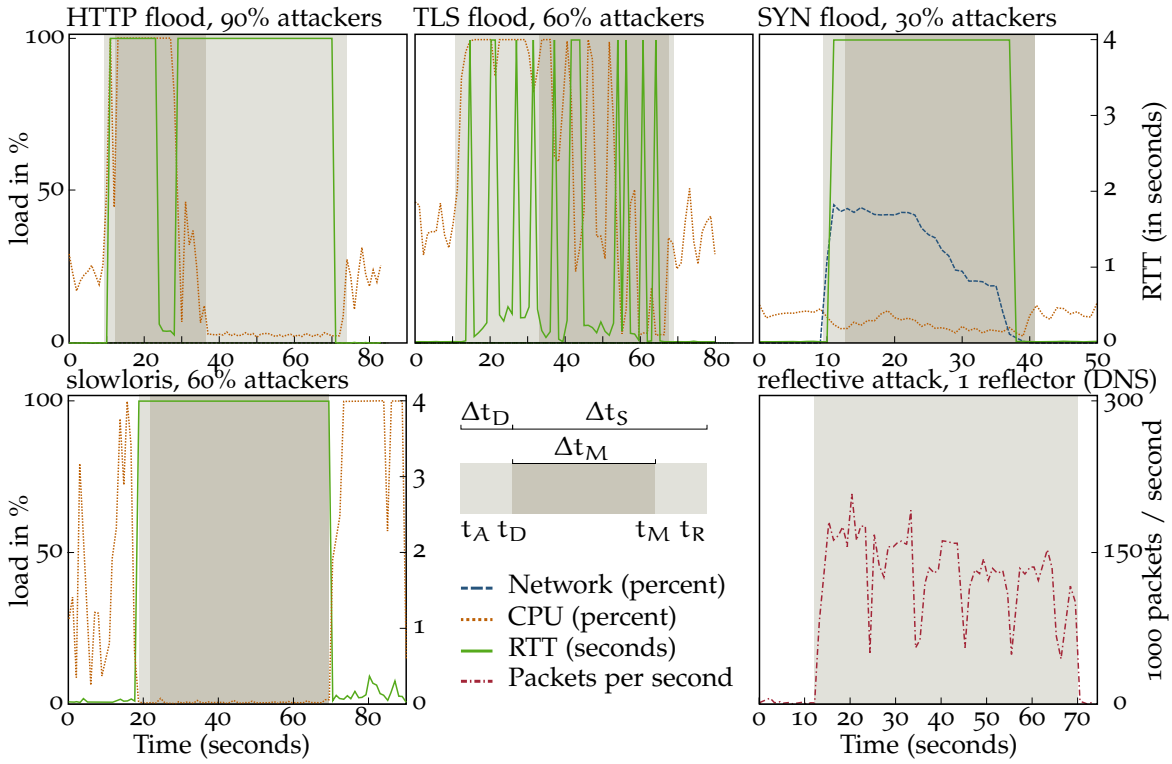


Figure 37: Example test runs of the different attacks. Partially based on [34].

the the recovery process' speed primarily depends on the server configuration of the victim. For example, the host operator could restart the server to shorten the recovery time or change their configuration to optimize recovery. However, this would mean that the host operator would take part in the mitigation effort, which is not part of the scenario we examine.

Figure 37 shows representative example test runs of attacks mitigated by the system, highlighting the difference between the attacks. For example, a SYN-flooding attack shown in the top right only takes effect after the TCP connection table of the web server is filled. After that moment, however, the server is completely unreachable. CPU load goes down as no connection request comes through. In contrast to the other two flooding attacks and the slow attack, a significant amount of traffic in the network can be observed, taking up more than 40 % of the available bandwidth, which corresponds to 4 Gbit/s. The RTT suddenly goes up to infinity, meaning an attack can be detected almost immediately. The scoring system works very well here, as single SYN packets with no corresponding ACK packet can easily be detected. As this is very unusual behavior for legitimate clients, the attackers can easily be distinguished from legitimate users. Blocking them immediately restores the availability of the server. As a small number of attackers is not enough to be an issue for the web server, the web server is already reachable before all attackers are

rerouted, and the mitigation system stops mitigating the attack. This restrained behavior also leads to less legitimate clients being rerouted in the 25 test runs. In this example run, 90 % of clients were attackers.

In the top left, Figure 37 shows the system's behavior during an HTTP flooding attack. The server is overloaded because it has to process a vast number of requests, which initially leads to a high CPU load, while network resources are not significantly affected. The attack only needs a few Mbit/s to be effective; therefore, traffic is not shown in the figure. After some time, disk I/O becomes the bottleneck, and the CPU load goes down. As soon as the RTT reaches the threshold set for the attack detection (1 s), the observation of the traffic and, therefore, the mitigation starts after four seconds. The highest-rated clients according to the suspiciousness score are blocked. At $t_M = 37$ s, the last offender is forwarded to the CAPTCHA server. The web server still needs time to recover from the attack — until t_R, 63 seconds after the beginning of the attack. CPU load goes up as soon as the server can handle new requests. Meanwhile, regular clients are also forced to fill out the CAPTCHAs, as the mitigation system observes that the server is still not reachable. However, the legitimate clients would not be able to access the data on the server before recovery regardless, since it is not available yet. Therefore, being forced to fill out a CAPTCHA can be seen as a mild annoyance but not as a serious issue.

In the top middle, Figure 37 shows a typical attack run and mitigation of the TLS flooding attack. The RTT highly fluctuates in this run, which in turn leads to a comparably late detection. However, as soon as the attack is mitigated, the server is reachable nearly immediately. Network resource consumption is similarly low to HTTP flooding.

The slowloris attack in the lower-left corner shows near-immediate recognition (as soon as the server is unavailable). The mitigation effort takes some time as it only becomes effective when most clients are blocked and can be optimized with the specialized identification scheme evaluated separately.

The attack run of the reflective attack in the lower-right corner is very different from the other attack runs. As the attack targets the network and not the server directly, the metric we observe here is merely the packets per second sent through the switch. With the attack overloading the switch itself, the detection takes quite some time, nearly 60 s as the switch is incapable of handling the traffic and fails to send its observations earlier. However, as soon as the attack is detected, the specialized mitigation works instantly.

These are only a few example runs out of the hundreds of test runs that demonstrate the attack and mitigation behavior. Figure 38 illustrates the results of the overall analysis of the flooding attacks. On the left, it shows the detection time of the various attacks. The highly fluctuating RTT of the TLS flooding attack affects the detection time

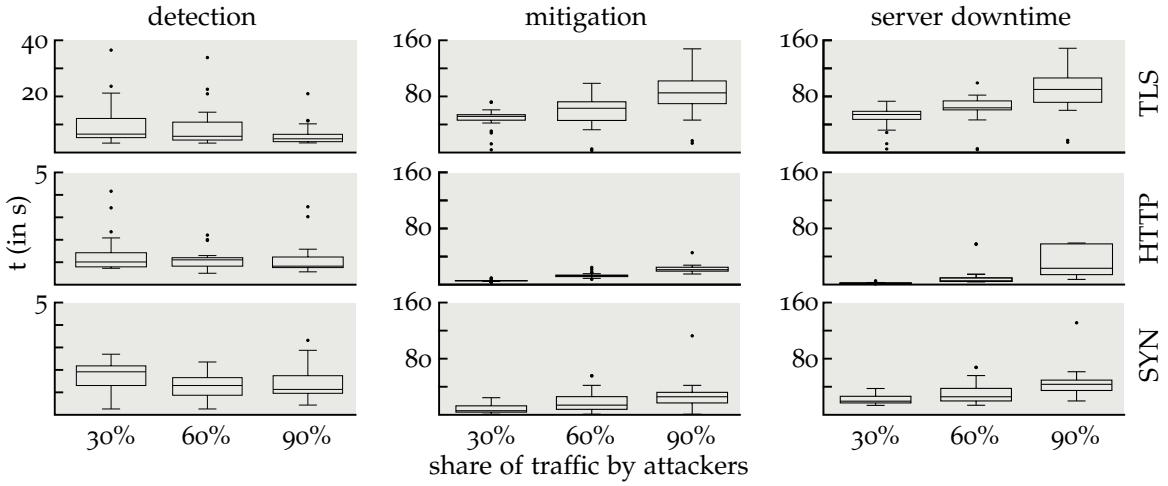


Figure 38: Detection time, mitigation time, and server downtime for the flooding attacks [4].

for this attack. Detecting a TLS flooding attack takes twice as much time as detecting the other two types of attacks as the server is sporadically available throughout the attack run. Another trend that can be observed is that detection is faster for stronger attacks. The mitigation time Δt_M is dependent on attack type and strength. SYN flooding and HTTP flooding attacks are mitigated within roughly half a minute while — depending on strength — the mitigation of TLS flooding can take 90 seconds or more. The server downtime for each attack type and attack strength is shown on the right of the figure. The difference between mitigation time and server downtime is only noticeable with HTTP flooding attacks where the recovery phase can take up to one minute.

As previously mentioned, it can be observed that some regular clients are forwarded to the CAPTCHA server. During our tests, of the 25 measurements for each scenario, this happened between 0 and 3 times, with the exception of the HTTP test with 90 % attackers. Here, in 13 cases, clients were falsely redirected with lead to a median of 18 % of all legitimate clients being redirected, which can be attributed to the rather long recovery time. With SYN flooding, falsely redirected clients could only be found during one test with 90 % attackers, where 5 % of the legitimate clients were redirected.

For the slow attacks, the falsely redirected clients are between 20 % and up to 75 % showing that the general identification scheme does not work well for the slow attack scenario, justifying a specialized identification scheme.

Conclusion

Overall, the performance evaluation shows overall satisfying results for the application-layer flooding attacks (HTTP flooding and TLS flooding). The SYN flooding mitigation also works great under test

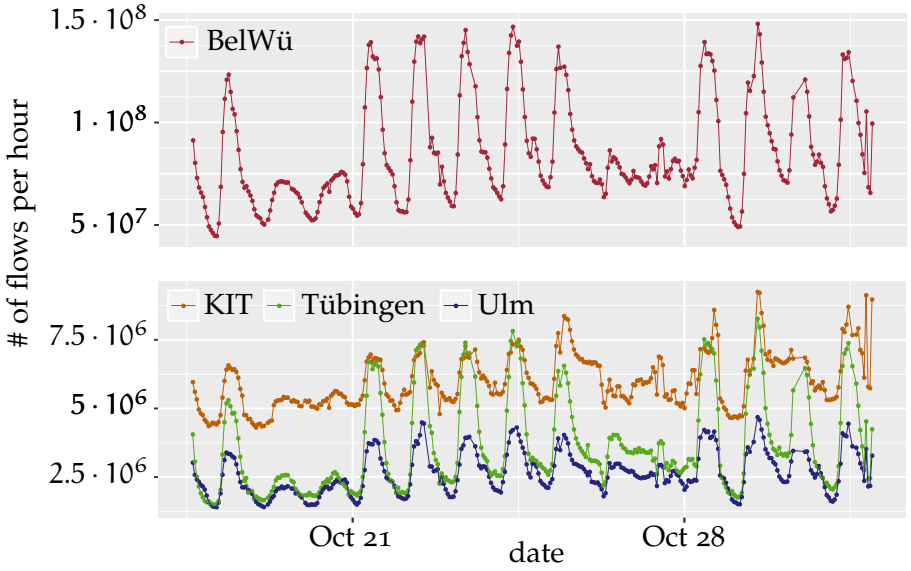


Figure 39: Recorded flows per hour in the BelWü data set for the full network (top) and the university subnets. The three university subnets are only a small part of the full data set.

conditions. However, the switch was still able to handle all SYN flooding traffic. In case of a stronger attack that overloads the bottleneck, other mitigation schemes need to be deployed. We recommend falling back to BGP blackhole routing in such a case. For reflective attacks, the mitigation worked perfectly, while the detection took far too long due to the overloaded switch confirming our assumption that a specialized detection method is needed. Detection and mitigation of slow attacks worked as expected, while bad identification results lead to long mitigation times and high false-positive rates, confirming our assumption that a specialized detection method is necessary.

11.6.2 *Detection Evaluation*

This section contains the evaluation of the detection mechanism based on network behavior, i. e., based on entropy measurements.

Thanks to the bwNetFlow project, we have access to live recordings of NetFlow traffic in the Baden-Württemberg Extended Lan research network. To analyze the applicability of our results to other networks, we analyzed the DDoS detection mechanism based on network behavior with recordings of other networks. Based on our analysis in Chapter 2, we chose the following benign data sets for this analysis:

- CAIDA 2019 This data set offers an extensive amount of data, recorded very recently within a small time frame of one hour. The CAIDA data set is divided into two parts entitled direction A and direction B, where the bulk of the flows is contained in direction B.

- Parts of the WIDE Day in the Life of the Internet recording of 2019. This data set contains a much longer recording of 48 hours. We chose two time periods of the data set for our analysis, namely two hours in the very early morning hours, to see how traffic behaves at night and two hours during peak utilization around noon. The size of the network (regarding the number of flows) is similar to the university networks in Baden-Württemberg.

- bwNetFlow recordings between October 17th and October 31st. A total of 343 hours were recorded. The recording program logs when it is overloaded and cannot record every flow it receives. This was the case during 16 hours of the recording, which were omitted in subsequent evaluations, leaving 327 hours.

These data sets were chosen as they are the most extensive data sets currently available that are — at the time of writing — also the latest and most contemporary data sets corroborating the claim of representativity. CAIDA and WIDE contain traffic recordings in the pcap file format. A transformation to NetFlow data was necessary first.

The bwNetFlow data set contains an ID for each flow, telling us at which institutions that flow originates or ends (depending on direction). With this ID, it is possible to analyze smaller subnets within the research network. For our evaluation, in addition to the full data set, we chose to analyze the networks of three universities in Baden-Württemberg, namely the Karlsruhe Institute of Technology, the University of Tübingen, and Ulm University as examples of networks with similar behavior but varying size. Figure 39 shows the number of flows per hour that are part of the data set. It contains a total of 27.6 billion flows (of those 25 billion TCP and 2 billion UDP; 2.6 % IPv6) from an average of 4 million hosts per hour. While Karlsruhe has a similar pattern to Ulm albeit a higher average flow rate, Tübingen alternates between the relatively low traffic volume at night similar to Ulm and the high traffic volume during the day similar to Karlsruhe.

Table 12 contains the average number of flows per hour for all data sets. When taking into account the lower sampling rate of the BW networks (recording only every 32nd flow while the other data sets contain all packets sent), the size of the BelWü network is similar to the CAIDA network (2.6 billion versus 1.6 billion flows). In comparison, the WIDE network is similar to the university networks (83 million to 192 million flows per hour versus 130 to 146 million flows per hour). While the BW and CAIDA networks are production networks with regular traffic patterns, the WIDE network mainly contains experimental traffic.

The documentations of all these data sets mention no attacks, and no typical attack patterns of DDoS attacks can be found. For the BW

Table 12: The data sets used in this evaluation. Flows are actually recorded flows, to determine actual network size for the BelWü networks, the sampling rate of $\frac{1}{32}$ has to be accounted for.

data set name	time span (local time)	average flows per hour
BelWü	2019-10-17 15:00 - 2019-11-01 00:00	84 423 579
BW-KIT	2019-10-17 15:00 - 2019-11-01 00:00	5 964 948
BW-Tübingen	2019-10-17 15:00 - 2019-11-01 00:00	3 586 561
BW-Ulm	2019-10-17 15:00 - 2019-11-01 00:00	2 631 059
CAIDA 2019 dir A	2019-01-17 13:00 - 2019-01-17 14:00	438 858 279
CAIDA 2019 dir B	2019-01-17 13:00 - 2019-01-17 14:00	1 135 740 233
WIDE Noon	2019-04-10 11:00 - 2019-04-10 13:00	145 573 794
WIDE Night	2019-04-08 01:00 - 2019-04-08 03:00	129 591 090

data sets, it was confirmed by operations of BelWü that there were no known DDoS attacks during the time of recording. Of course, as the data sets are actual recordings of production networks, there can be no guarantee that the data sets are attack free. However, we accept the lack of any sign of an attack within the data sets as a sufficient indicator. Unknown attacks within the data sets could mean that the classification of attacks versus no attacks would falsely classify a true positive as a false positive or a false negative as a true negative. With the extent of our analysis over several data sets and weeks, one missed attack would only have a small impact on the overall results.

To assure realistic attack patterns can be observers, we added known attack traffic to our analysis based on the known features of attacks. To analyze common features of attack data sets, we chose to analyze[8]:

- Booters containing 11 real attacks (7 DNS-based and 2 CharGen-based reflective attacks)[9].

8 a more extensive description of these data sets can be found in Chapter 2

9 https://www.simpleweb.org/w/index.php/Traces#Booters_-_An_analysis_of_DDoS-as-a-Service_Attacks

- USC/LANDER containing a simulated reflective DDoS attack using DNS servers as reflectors[10].

Based on these data sets and the associated documentation [271], we know that the start-up time of attacks is in the area of just a few seconds. Based on the *Imperva Threat Report*, we know, that attacks can be as short as 30 minutes while still being successful (i. e., the attack target is unreachable for hours). This means that swift detection is necessary. Therefore, we limited our evaluation to time frames of 30 s, 60 s, and 15 min, which results in attack detection times close to the same value (plus calculation time in the area of seconds) as anything beyond 15 min would impede a timely reaction.

Figure 40 shows the overview of all data sets under investigation and how the destination IP entropy changes over time. The CAIDA graphs show an evaluation of a period of one hour, WIDE two hours, and the BW data sets roughly two weeks. The red points are the measured values. All other points are hypothetical results during an attack. They were calculated by adding a certain number of flows to the calculation, all with the same destination IP (a pattern that can be observed during an attack). The number of additional flows is the *strength of the attack*, and here, it is defined dependent on the number of flows in the background traffic of the network. Therefore, an attack strength of 0.05 means that an additional 5 % of the flows were added to the calculation, all with the same destination IP address not present in the background traffic. The graphs show values calculated for periods of 30 s (CAIDA and WIDE), and 15 min (BelWü data sets). The actual values of the entropies cannot be compared between the graphs as different periods were used for the calculation. In the BelWü data sets, the missing hours can clearly be seen as missing data points in these graphs.

An observation that can be made solely based on these graphs is that bigger networks tend to be more stable (see the difference between CAIDA direction A and B; even more so between CAIDA and WIDE), which seems to lead to a more distinct pattern during an attack versus no attack. However, a 5 % attack strength in a bigger network also means that the attack is much stronger.

Looking at the BelWü graphs, compared to Figure 39, the destination IP entropy seems to be far more stable than the number of flows in the network. This could indicate that the entropy is a far better metric to detect anomalies than the number of flows.

Figure 41 shows a similar collection of graphs. However, here, we observe the source port entropy in the networks. As we calculate this to determine whether the attack is a reflective attack based on UDP,

10 Scrambled Internet Trace Measurement dataset, IMPACT ID: USC-LANDER/DoS_DNS_amplification-20130617/rev5529. Traces taken 2013-06-17 to 2013-06-17. Provided by the USC/LANDER project (https://www.isi.edu/ant/lander).

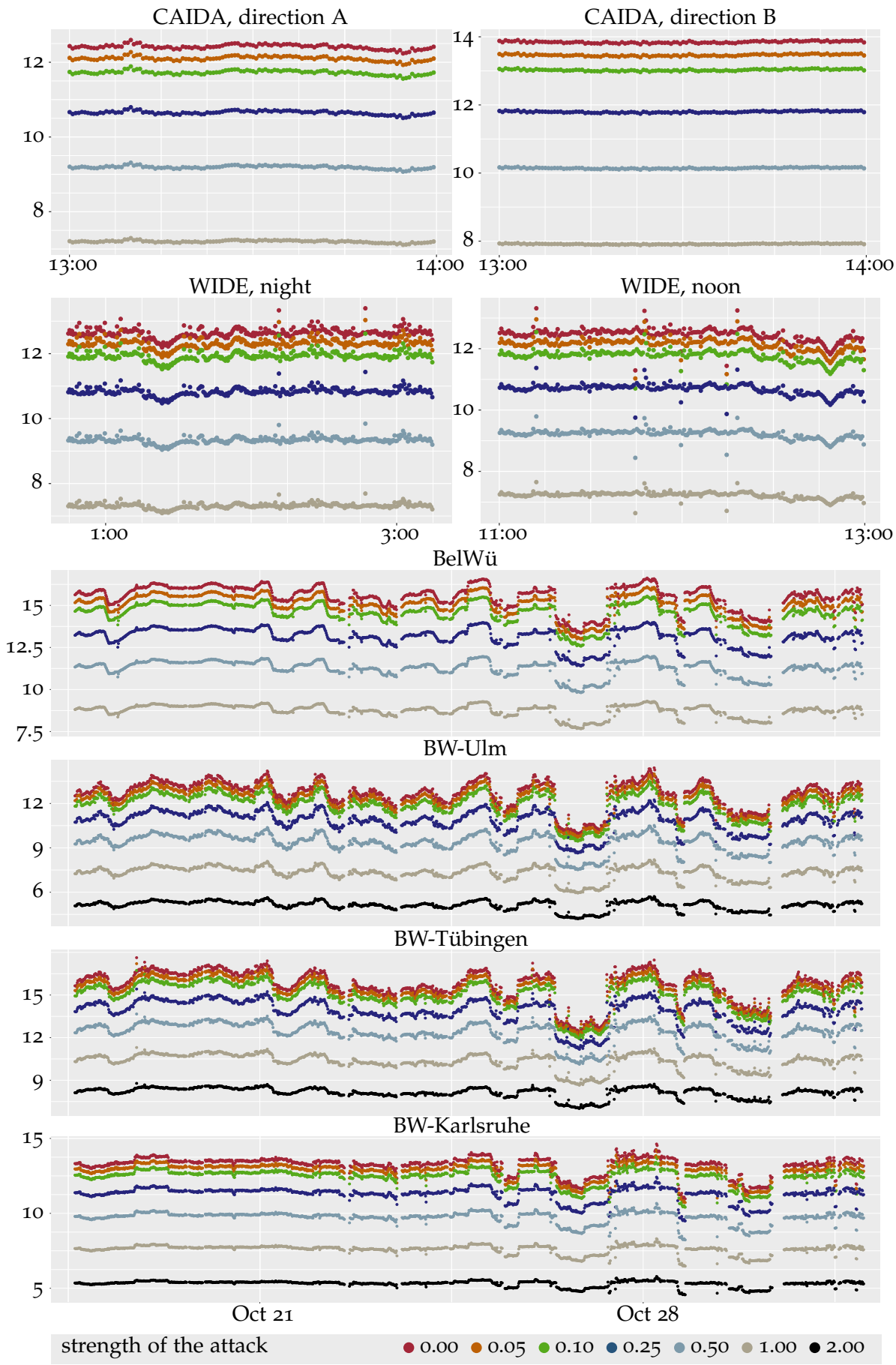


Figure 40: Changes of the destination IP entropy over time in all data sets in the presence of attacks.

the source port entropy is calculated based only on the UDP flows in the respective networks.

Due to the far lower number of flows analyzed, we can observe more variation in the network; the source port entropy of UDP flows seems to be far less stable than the destination IP entropy of all flows.

In the following, receiver operating characteristics (ROC) curves are used to show the diagnostic abilities of the entropy calculations as a binary classifier (i. e., attack versus no attack). On the x-axis, these graphs show the true-positive rate, i. e., the number of attacks found over the number of attacks in the network. The y-axis shows the true-negative rate, which is the number of negatives (i. e., no attack) found over the number of negatives in the network. Setting up the graph like this means that a value in the top-right corner represents a perfect classification while the lower-left corner would mean the inverse, the lower-right corner would mean that all values are classified as positives, and the lop-left corner that all values are classified as negatives. As a base for this calculation, we take the values shown in Figures 40 and 41. For each time step, we calculate whether the hypothetical attack with strength α would be detected based on our scheme described in Section 11.2. For this calculation, we alter the threshold in small steps between two extreme values, resulting in the curve that can be seen in the graphs.

Figure 42 shows the resulting curves for the BelWü data set, analyzing every 2048th flow. The three graphs use different epoch lengths, i. e., lengths of the time steps used to calculate the entropies. As can be seen in the graphs for 30 s, 60 s, and 15 min, the difference is small. However, 60 s has slightly better results than the other two with 30 s as a close second. We decided to continue all other evaluations based on the 30 s epoch length because, in our view, detecting attacks earlier outweighs the very small advantage of the longer epoch time of 60 s.

The figures used in the following to corroborate our findings are only a small part of an extensive evaluation. An extensive set of graphs showing how well attacks can be found in the data sets under investigation can be seen in the Appendix in Figures 49 to 53. The figures in the Appendix contain ROC curves of all data sets used in our analysis for both destination IP entropy and source port entropy at different sampling rates.

Network monitoring usually does not analyze every packet and flow — especially in networks with high throughput. In these networks, flows are sampled, and only a small part of the flows are used for analysis. For example, the BelWü network we are analyzing here. Only every 32nd flow is analyzed in this network. This is due to the fact that the Cisco routers BelWü uses are not capable of extracting more flows. Many other observation systems only allow much lower sampling rates, analyzing only every 1 000th, or 10 000th flow. There-

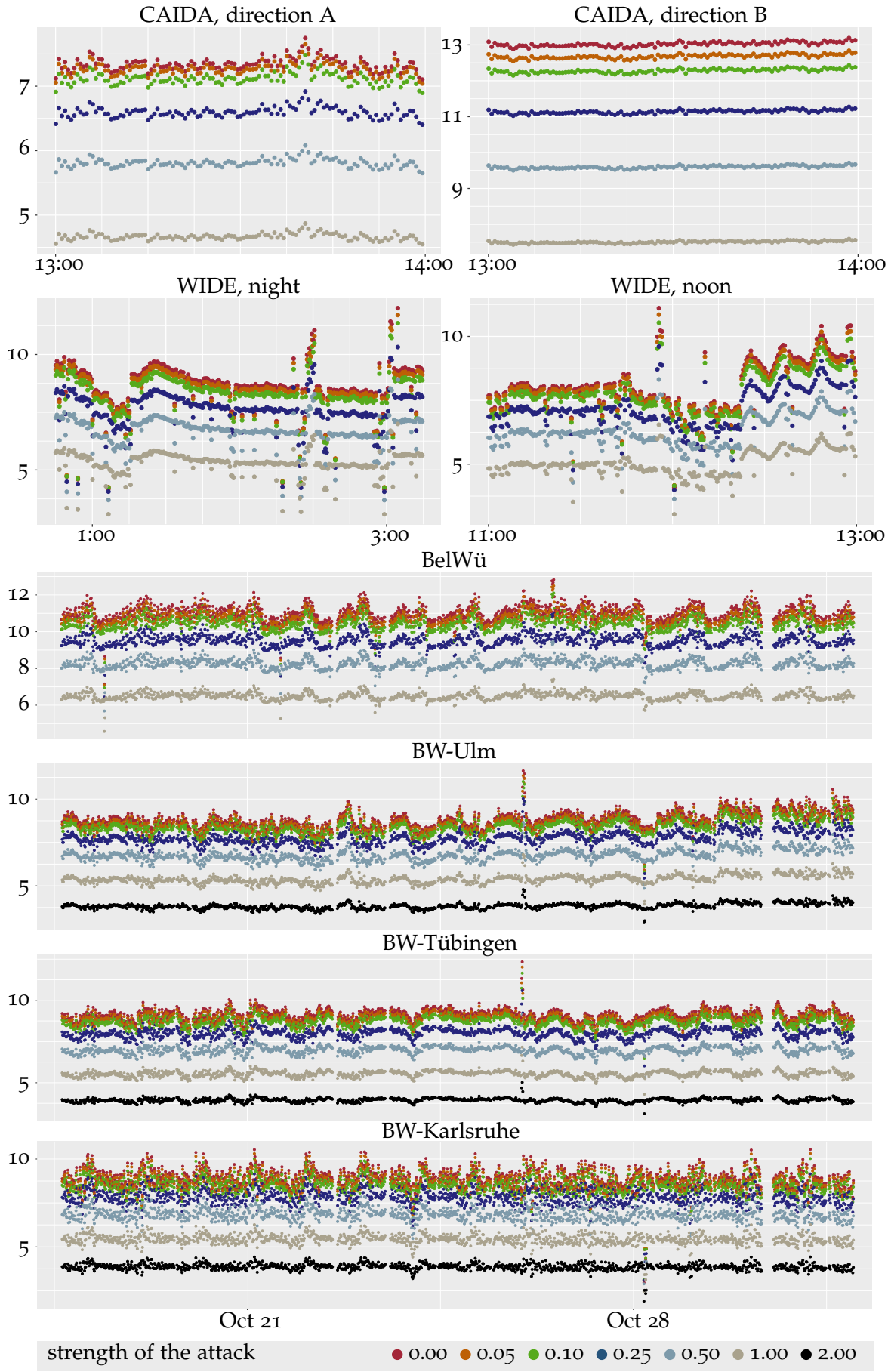


Figure 41: Changes of the source port entropy over time in all data sets in the presence of attacks.

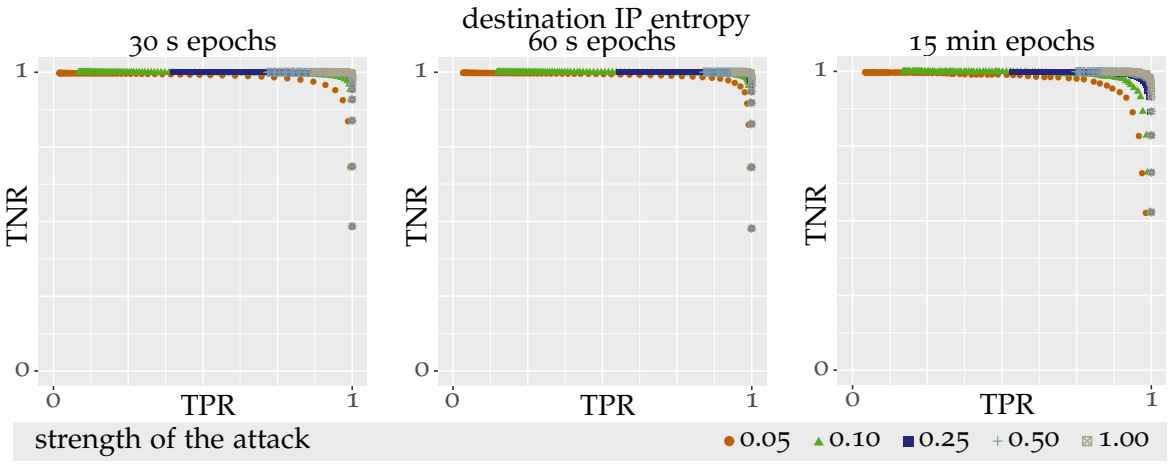


Figure 42: Comparison of different epoch lengths as a base for the entropy calculations. Calculations done based on the 327 hours BelWü recording.

fore, it is important to evaluate the impact of a low sampling rate on the quality of the detection mechanisms.

Figure 43 contains ROC curves for both destination IP entropy and source port entropy of the CAIDA direction B data set at different sampling rates. The smaller data set of the source port entropy calculations (only using UDP flows) is affected a lot earlier at a higher sampling rate than the destination IP entropy calculation. The destination IP entropy calculation shows similar behavior when analyzing every 65 536th flow as the source port entropy when analyzing every 2 048th flow.

A similar observation can be made when comparing similar networks of different sizes. The BelWü network and the university networks in BelWü show similar traffic compositions. However, as can be seen in Figure 44, the attack classification works a lot better in the bigger network. In this graph, every 128th flow was analyzed.

The evaluation shows that the destination IP entropy can be used to detect DDoS attacks reliably. In addition, the source port entropy can be used to determine whether the DDoS attack under investigation is a reflective DDoS attack which allows targeted defense mechanisms to function reliably. Depending on the size of the network and the size of attacks that the operator wants to detect in the network, specific thresholds need to be set for the two classifications. Sampling can be used to minimize calculation overhead. However, if the network naturally shows high variations in the entropies, lowering the sampling rate could lead to more reliable results. While we only observed a few networks, our analysis indicates that the quality of the classification solely relies on the size of the network (measured by the number of flows) and the sampling rate. Networks with similar sizes showed similar behavior. However, all networks under observation are somewhat similar in traffic composition (e. g., share of UDP), and all data sets are recorded in networks primarily used by researchers

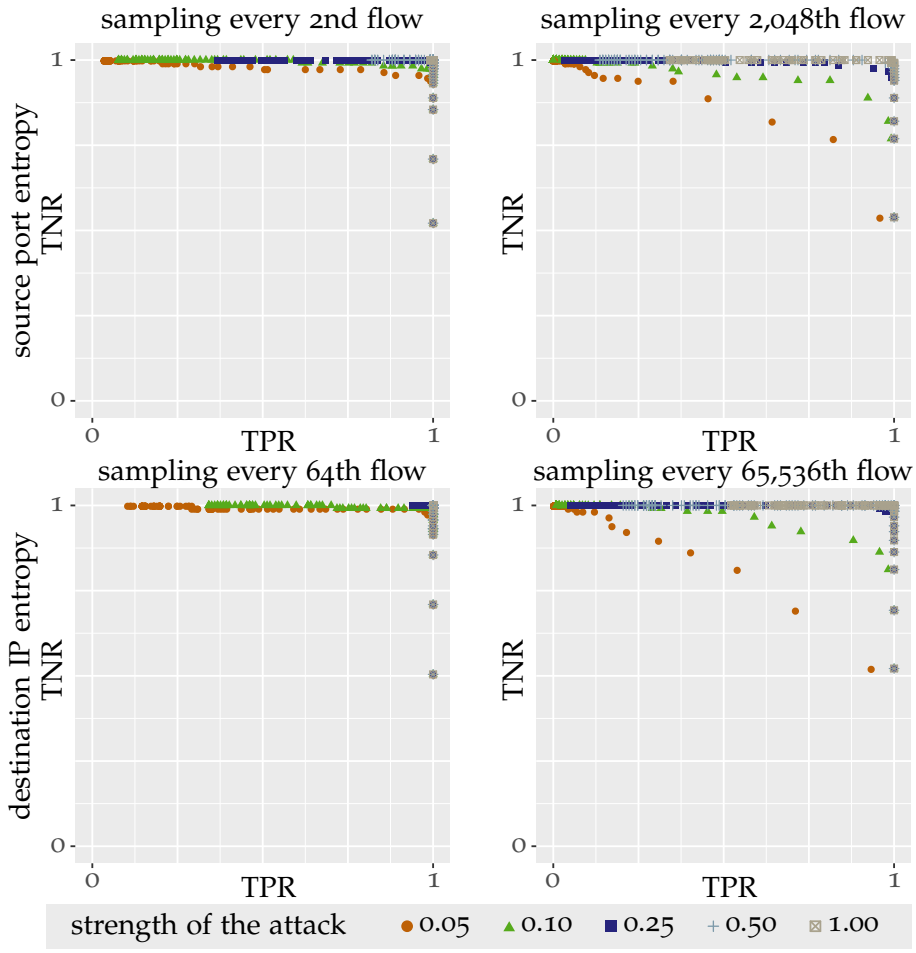


Figure 43: Effect of different sampling rates on the classification performance. Shown here on the example of the CAIDA direction B data set.

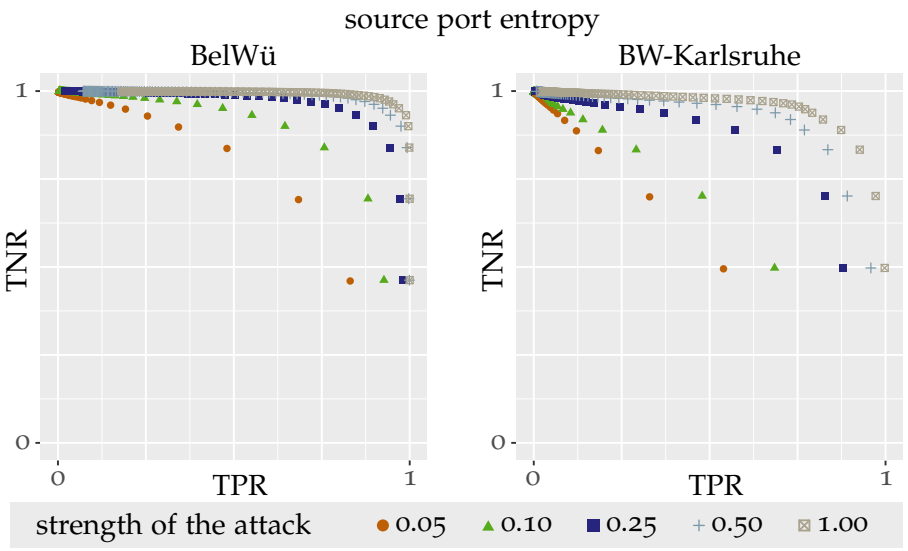


Figure 44: Source port entropy in two networks of similar traffic compositions but different sizes. Both graphs show the analysis of 327 hours. Every 128th flow was analzed.

and students both privately and at work. More research would be needed to see if this holds true for other network types.

11.6.3 *Identification Evaluation*

In the following, we present the evaluation results of the specialized identification of slow attack clients. For this, we evaluate the schemes presented earlier: long Connections (LC), low packet rate (LPR), packet distance uniformity (PDU), the combination of LPR and PDU (LPR-PDU), low mean packet rate (MPR), and low packet rate variance (PRV). We describe the setup and workloads based on real-life traffic scenarios. We then present the results and discuss their implications.

Data Set

For this evaluation, a special network data set was necessary. While our previous evaluations observed general networks and their traffic, we now look into specific traffic to and from a web server. Therefore, we used the SUEE data sets introduced in Chapter 5. They were recorded at two different times at a web server located at Ulm University. SUEE1 contains one day of traffic and can be used as a training data set. SUEE8 contains eight days of traffic and can then, in turn, be used as a testing data set. The server administrator did not observe any attacks on their server in the two time frames, and the server was available for the full time of the recordings. No other suspicious activity was observed neither live nor by manually analyzing the traffic traces.

Attacker Model

The attackers in these scenarios have access to a large number of distributed network resources (e. g., a botnet). Attackers making use of DDoS attacks are differing greatly in resources and technical knowledge. We differentiate between two types of attackers. On the one hand, the simple attacker. The simple attacker uses tools that are readily available, and several of those exist for slow attacks. We also make use of these very tools and the options they provide to emulate the behavior of this attack type. The second attacker model features a more sophisticated attacker that knows how the attacks work, understands how these attacks are usually mitigated, and built their own or made adjustments to the attacking tool to circumvent detection.

Implementation

We use the Zeek-based framework previously discussed that was also used for the evaluation of the flooding attack mitigation. Zeek al-

ready offers some of the metrics necessary for the analysis, such as the packet rate as built-in functions. Other metrics, such as the packet rate without the TCP handshake, have to be calculated without a built-in function of Zeek. This can lead to more processing effort for these schemes as the built-in functions have been potentially optimized over the years, while the newly added features might be less efficient.

Attackers for the first attack scenario are simulated with the tools slowloris 0.1.4[11] for slow header attacks and slowHTTPTest 1.6[12] for slow body attacks.The attacking tools were adapted to allow IP spoofing to simulate distributed attacks and are left in standard configuration apart from that. The parameters for slowHTTPTest are 30 seconds intervals, 8 192 bytes for the Content-length header, 10 bytes POST-body length per packet, and one socket per client. Slowloris is also configured to use only one socket per client. The default configuration is left in place in all other settings, resulting in a packet interval of 15 seconds.

For the second attack scenario, we modified the original slowloris tool and created a variant with less predictable behavior, called slowloris-ng[13]. Slowloris-ng includes several changes compared to the original slowloris. The additional features implement randomized behavior, which is configured to send in intervals of 15 seconds with a randomization interval of 5 seconds and sending the header lines as bursts of single messages per character. This tool shows how applicable the presented schemes are for improved attackers compared to easily accessible attacking scripts.

For each of these tools, 49 to 50 clients[14] are started simultaneously with different IP addresses to attack the web server. These attacks are run in parallel with the benign recordings. Due to the vast differences between the attacks, we choose to determine the best thresholds for each of the three attacks separately and evaluate these thresholds against all attacks.

Each of our schemes can be configured with specific parameters — for example, for the long connection scheme, a threshold defines after how many seconds a connection is considered suspicious. For other schemes — such as low packet rate — the optimal rate needs to be determined. The optimal values are extracted by testing the behavior of the framework with SUEE1, the quality of the classification then is evaluated on SUEE8. The number of suspicious packets per client (number of strikes) necessary to deem a client an attacker is another parameter under investigation. Its purpose is to reduce false positives when a benign client sends only one packet or a very small number

11 https://github.com/gkbrk/slowloris

12 https://github.com/shekyan/slowhttptest

13 https://github.com/vs-uulm/slowloris-ng

14 The different number of clients stems from a bug in the SUEE data sets where some clients using different attacks had the same IP addresses and had to be ignored.

of packets that incidentally fall below the threshold. However, this should have an impact on the identification time. During our preliminary tests, we have noticed that these schemes (except for LC) behave differently depending on whether the TCP handshake is taken into account. Therefore, we evaluate these with and without measuring the handshake packets.

The evaluation thresholds are determined by testing the mitigation system with the SUEE1 data set with induced attacks of each type. The number of strikes for detection is set to one. The thresholds are found using the bisection method, starting with two extreme values that would result in the detection of all clients — benign and attackers — and detection of no clients respectively. The balanced accuracy is used as a quality metric. Balanced accuracy (BACC) is defined as $BACC = (\frac{TP}{TP+FN} + \frac{TN}{TN+FP}) \cdot 0.5$ with the true positive values TP, false positive FP, true negative TN, and false negative FN. We have decided to use balanced accuracy over accuracy, as it takes the unbalance of the data sets into account (only 49 to 50 attackers versus up to 8 286 benign clients). Otherwise, a completely worthless classifier that classifies everything as benign would result in an accuracy of up to 0.994, while the balanced accuracy would be 0.5, similar to a coin toss, resulting in a much more accurate indicator. Optimized balanced accuracy has the advantage of resulting in one clear value, that can be taken as a reasonable estimation of the quality of a scheme.

Results

The thresholds determined by this test can be seen in Table 13. The table shows that some schemes are very similar for all attacks (e. g., MPR, PRV; LC for slowloris and slowHTTPTest) while other schemes show big differences for different attacks (e. g., LPR-PDU). MPR and PRV show extreme differences depending on if the TCP handshake is part of the evaluation or not. The high thresholds when ignoring the handshake might imply that these schemes might not be applicable without the TCP handshake. For LPR-PDU, we also evaluate the maximum values for each partial scheme (highlighted in the table in bold) in addition to the best thresholds for each attack.

For every classification scheme, there are two things to consider. On the one hand, how precise the identification is of each scheme for each attack, measuring if the attack can be mitigated successfully without too many blocked benign clients. Again, we use the balanced accuracy to assess the quality of the scheme but report all false/true positive and false/true negative values as well. Furthermore, another very important aspect is the identification time, i. e., the mean time each attacker remains unidentified. Slow attacks work by opening as many connections as possible and keeping them open as long as possible to ensure maximum impact. If all attackers can be identified correctly, but the identification time is too high, the attack might still

Table 13: Overview of the ideal thresholds for each scheme and attack for data set SUEE1.

schemes	TCP handshake	slowloris	slowHTTPTest	slowloris-ng
LC		$d = 2.1e{-}5s$	$d = 2.1e{-}5s$	$d = 0.0999727s$
LPR	yes	$p = 0.091756Hz$	$p = 0.01739Hz$	$p = 0.783869Hz$
	no	$p = 0.079935Hz$	$p = 0.03806Hz$	$p = 0.77687Hz$
PDU	yes	$\Delta = 5.9e{-}5s$	$\Delta = 2.5e{-}5s$	$\Delta = 2.5e{-}5s$
	no	$\Delta = 1.4e{-}5s$	$\Delta = 0.000631s$	$\Delta = 1e{-}6s$
LPR-PDU	yes	$p = 0.091756Hz$	$p = 0.01739Hz$	$\mathbf{p = 0.783869Hz}$
		$\Delta = \mathbf{5.9e{-}5s}$	$\Delta = 2.5e{-}5s$	$\Delta = 4.1e{-}5s$
	no	$p = 0.079935Hz$	$p = 0.03806Hz$	$\mathbf{p = 0.77687Hz}$
		$\Delta = 1.4e{-}5s$	$\Delta = \mathbf{0.000631s}$	$\Delta = 1e{-}6s$
MPR	yes	$\bar{p} = 0.83315Hz$	$\bar{p} = 0.83315Hz$	$\bar{p} = 0.83315Hz$
	no	$\bar{p} = 4049Hz$	$\bar{p} = 21845Hz$	$\bar{p} = 995Hz$
PRV	yes	$\sigma^2 = 0.028007Hz^2$	$\sigma^2 = 0.028007Hz^2$	$\sigma^2 = 0.028007Hz^2$
	no	$\sigma^2 = 1332kHz^2$	$\sigma^2 = 1332kHz^2$	$\sigma^2 = 1332kHz^2$

be successful. It might even be worth trading accuracy for lower identification times if necessary.

Table 14 shows the results for the schemes. The long connection scheme (LC) can be used to detect slowHTTPTest fast with a highly varying false positive rate between 3.4 %, and 58 %. It cannot be used to detect the other attacks. For slowloris-ng, it performs on the same level as a coin toss. The low mean packet rate scheme (MPR) and low packet rate variance scheme (PRV) show similar results. They can detect slowHTTPTest but are close to useless for the other attacks. For these three schemes, the TCP handshake has to be taken into account. The table also contains our results for the schemes low packet rate (LPR) and packet distance uniformity (PDU). The results show that these metrics can be used to detect attacks (except for rare cases, all attackers were found); however, the false-positive rate varies significantly.

Low packet rate is a good classifier for the basic attacks slowloris and slowHTTPTest with a balanced accuracy of 0.96 to 0.98 for the SUEE8 data set. However, identification times of up to 210 seconds per client have to be considered. Packet distance uniformity is much faster but also less reliable than LPR for the basic attacks; it performs better than LPR when faced with the improved slowloris-ng attack.

Table 14: Evaluation results dependent on scheme, whether or not TCP handshake is evaluated, attack (SL: slowloris, SH: slowHTTPTest, NG: slowloris-ng), using ideal thresholds; *identification times* are based on the true positives in seconds.

schemes	TCP handshake	attack	true positive	false positive	false negative	true negative	balanced accuracy	identification time in s
LC		SL	32	4959	17	3544	0.535	$\bar{t} = 0.84\ \sigma = 1.35$
		SH	49	4959	0	3544	0.708	$\bar{t} = 12.86\ \sigma = 8.81$
		NG	50	8502	0	1	0.5	$\bar{t} = 0.12\ \sigma = 0.59$
MPR	N	SL	49	7690	0	813	0.548	$\bar{t} = 2.61\ \sigma = 5.10$
		SH	49	7690	0	813	0.548	$\bar{t} = 24.34\ \sigma = 37.56$
		NG	50	7690	0	813	0.548	$\bar{t} = 4.39\ \sigma = 6.68$
	Y	SL	19	611	30	7892	0.658	$\bar{t} = 71.81\ \sigma = 42.96$
		SH	49	611	0	7892	0.964	$\bar{t} = 108.73\ \sigma = 52.65$
		NG	14	611	36	7892	0.604	$\bar{t} = 106.34\ \sigma = 85.91$
PRV	N	SL	49	7691	0	812	0.548	$\bar{t} = 1.07\ \sigma = 1.36$
		SH	49	7691	0	812	0.548	$\bar{t} = 20.58\ \sigma = 38.77$
		NG	50	7691	0	812	0.548	$\bar{t} = 1.60\ \sigma = 1.50$
	Y	SL	24	1431	25	7072	0.661	$\bar{t} = 1.46\ \sigma = 1.19$
		SH	49	1431	0	7072	0.916	$\bar{t} = 82.18\ \sigma = 44.05$
		NG	13	1431	37	7072	0.546	$\bar{t} = 2.00\ \sigma = 0.00$
LPR	N	SL	49	641	0	7862	0.962	$\bar{t} = 211.26\ \sigma = 28.65$
		SH	49	403	0	8100	0.976	$\bar{t} = 210.21\ \sigma = 28.65$
		NG	50	3853	0	4650	0.773	$\bar{t} = 52.87\ \sigma = 51.39$
	Y	SL	49	1019	0	7484	0.94	$\bar{t} = 174.83\ \sigma = 34.78$
		SH	49	139	0	8364	0.992	$\bar{t} = 240.06\ \sigma = 0.07$
		NG	50	4242	0	4261	0.751	$\bar{t} = 38.77\ \sigma = 55.31$
PDU	N	SL	49	1884	0	6619	0.889	$\bar{t} = 46.09\ \sigma = 37.79$
		SH	49	3502	0	5001	0.794	$\bar{t} = 105.63\ \sigma = 42.73$
		NG	50	538	0	7965	0.968	$\bar{t} = 12.22\ \sigma = 14.01$
	Y	SL	49	4021	0	4482	0.764	$\bar{t} = 5.94\ \sigma = 33.78$
		SH	49	3407	0	5096	0.8	$\bar{t} = 5.88\ \sigma = 6.69$
		NG	49	3407	1	5096	0.79	$\bar{t} = 1.56\ \sigma = 1.49$
LPR-PDU	N	SL	49	217	0	8286	0.987	$\bar{t} = 211.26\ \sigma = 28.65$
		SH	49	197	0	8306	0.988	$\bar{t} = 210.21\ \sigma = 0.01$
		NG	50	315	0	8188	0.981	$\bar{t} = 55.85\ \sigma = 50.06$
	Y	SL	49	471	0	8032	0.972	$\bar{t} = 176.36\ \sigma = 35.93$
		SH	49	88	0	8415	0.995	$\bar{t} = 240.06\ \sigma = 0.07$
		NG	50	1509	0	6994	0.911	$\bar{t} = 39.21\ \sigma = 55.03$

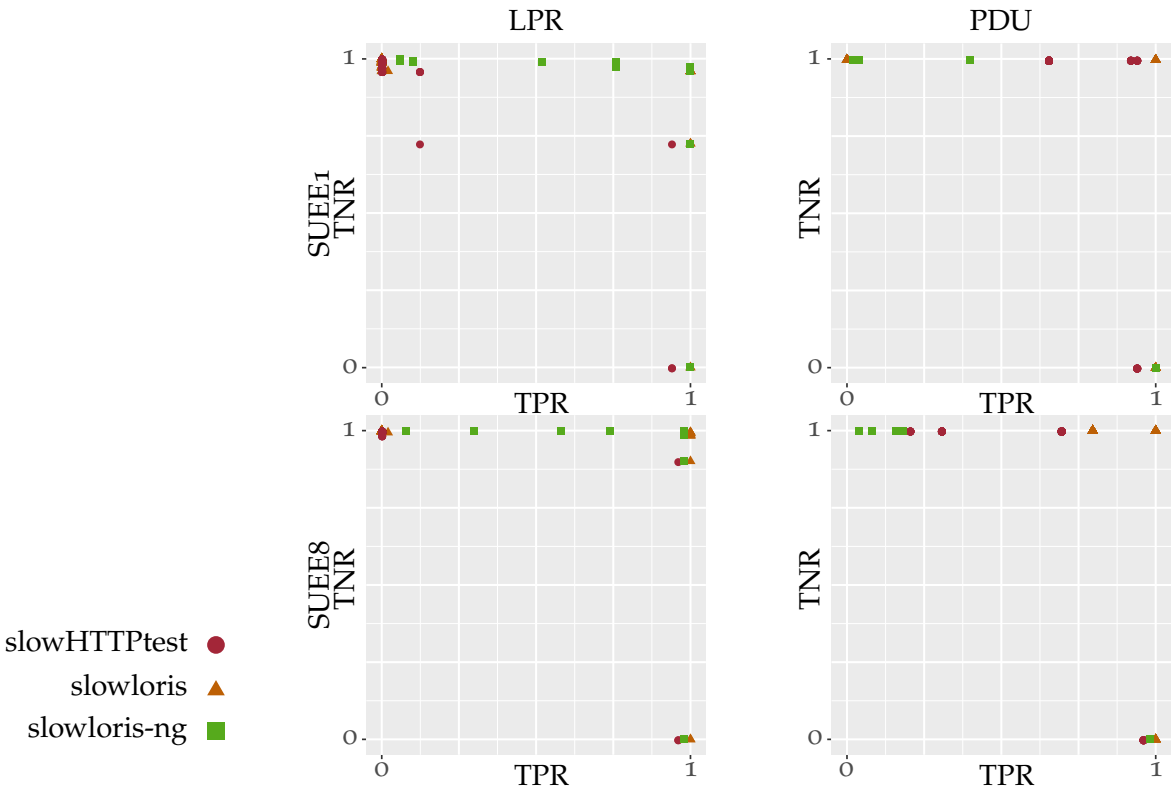


Figure 45: ROC curves for LPR and PDU for the three slow attack types in the two SUEE data sets.

The results show that only LPR and PDU are promising identification schemes and are analyzed further. Figure 45 shows receiver operating characteristics (ROC) curves for the two data sets SUEE1 and SUEE8 for LPR and PDU for all three attack types. The graphs show that LPR is a very good metric to find slowloris and slowloris-ng. However, it has problems discerning slowHTTPtest traffic and benign traffic. Here, PDU does a much better job while it cannot detect slowloris-ng, showing that the changes made to slowloris work very well to avoid detection. The points in theses graphs are very distant from each other. This is because minor changes of the threshold often affect the classification of a large part of the flows or none at all (many points in the graphs overlap) as many flows share the same features. That means that thresholds have to be chosen very carefully to avoid wrong classifications. All in all, these tests show that neither LPR nor PDU is able to identify the attackers fully reliably on their own.

The AND-combination of the aforementioned schemes (LPR-PDU) shows much better results than the two schemes, each alone. With a balanced accuracy of up to 0.987 without and 0.995 with TCP handshake, this method proves to be the most reliable scheme. However, as a combined method, it also inherits the high identification time of LPR with the best threshold pairs for each attack.

Table 15: Evaluation results for LPR-PDU when for each partial scheme the maximum threshold is chosen (bold values in Table 13).

TCP handshake	attack	true positive	false positive	false negative	true negative	balanced accuracy	identification time in s
N	SL	49	1261	0	7242	0.926	$\bar{t} = 20.35\ \sigma = 11.28$
	SH	49	1261	0	7242	0.926	$\bar{t} = 107.47\ \sigma = 38.59$
	NG	50	1261	0	7242	0.926	$\bar{t} = 52.87\ \sigma = 51.39$
Y	SL	49	1603	0	6900	0.906	$\bar{t} = 14.83\ \sigma = 33.02$
	SH	49	1603	0	6900	0.906	$\bar{t} = 13.05\ \sigma = 7.75$
	NG	50	1603	0	6900	0.906	$\bar{t} = 39.21\ \sigma = 55.03$

Up to here, we evaluated whether the schemes can work with the right thresholds for each attack. However, when defending a real network, we do not know which attack the attacker will choose. For the most promising schemes (LPR, PDU, and LPR-PDU), we therefore also evaluated how these schemes hold up when the threshold is not the ideal one for these attacks.

Figure 46 shows balanced accuracy results for LPR, PDU, and LPR-PDU, where for each threshold (or threshold pair), the diagrams show how good the classifiers are identifying the attackers in all three attacks. For LPR in the top two graphs, it can be seen that the best threshold for slowloris also works well against slowHTTPTest (with the same identification rate). The best threshold for slowHTTPTest, however, is unusable both for slowloris and slowloris-ng. As slowloris-ng has a higher packet rate than the other two attacks, their ideal threshold is much higher. This means more false positives and, therefore, lower balanced accuracy for all schemes. However, all attackers were detected reliably with this threshold.

For the packet distance uniformity scheme, when not taking the TCP handshake into account, the best threshold for slowloris also shows the same results for slowloris-ng. However, for slowHTTPTest, this value is unusable. In turn, the best threshold for slowloris-ng is unusable for the other two schemes. A packet distance of 0.000631s as the highest value results in more false positives than the other values but results in a true positive rate of 100 % for all attacks. When we include the TCP handshake, slowHTTPTest, and slowloris-ng work best with the same threshold, which is not usable for slowloris while the slowloris threshold results in a high false-positive rate for all attacks.

The combined scheme is evaluated with four different threshold pairs: The best pair for each attack and the maximum threshold values for each of these pairs (that can also be seen in Table 15). The

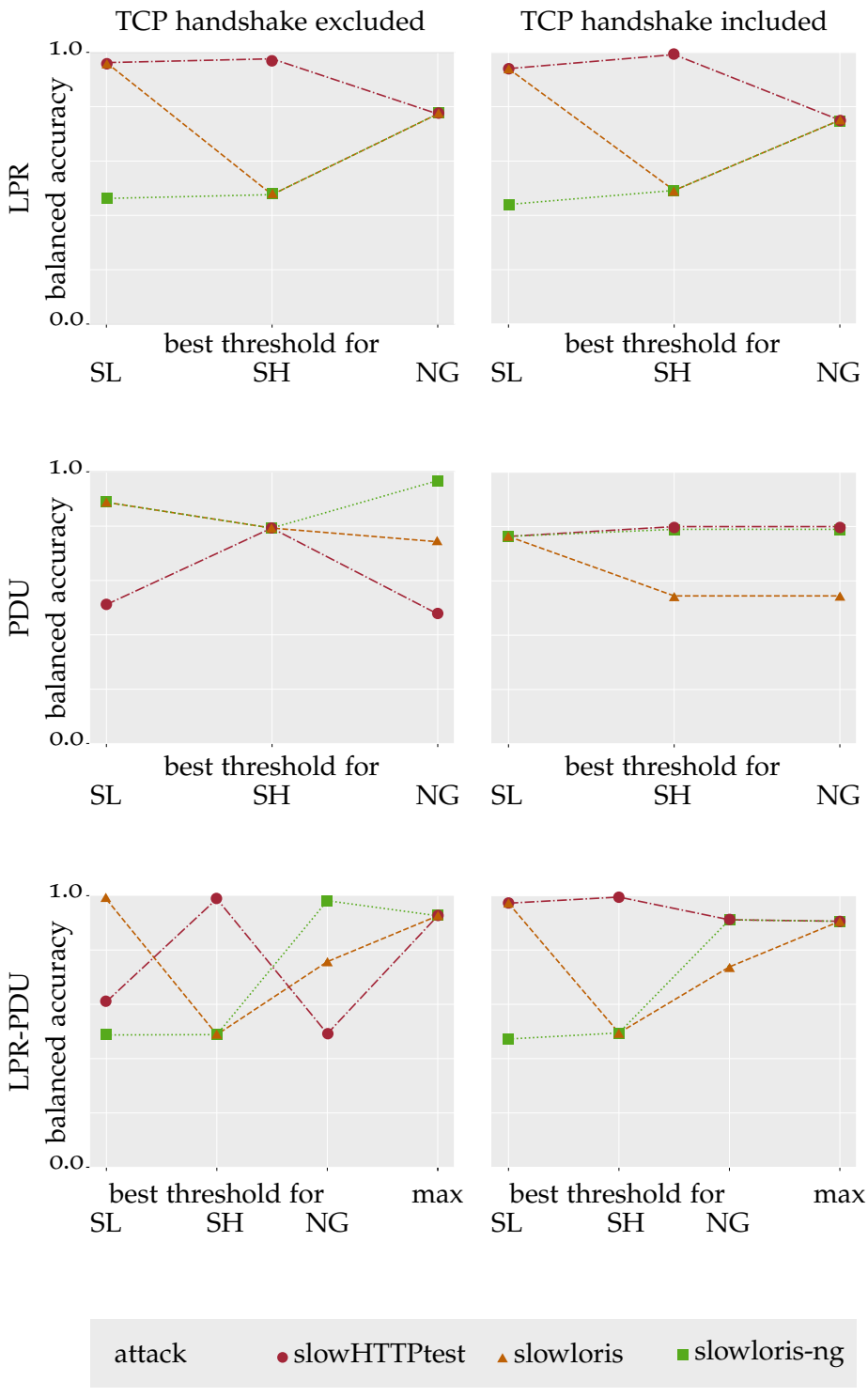


Figure 46: Balanced accuracy for LPR, PDU, and LPR-PDU (top to bottom) without TCP handshake and with TCP handshake (left to right). Each with the best thresholds for slowloris, slowHTTPTest, and slowloris-ng (left to right) and for LPR-PDU additionally the results when for each partial scheme threshold the maximum value of the three is used [5].

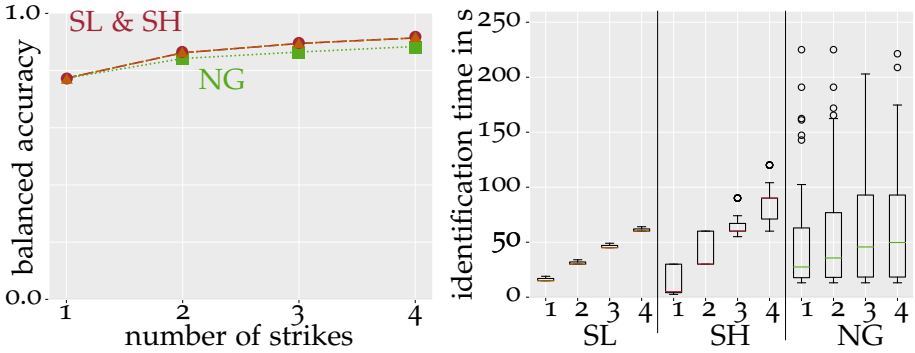


Figure 47: Evaluation results for balanced accuracy (left) and identification times (right) for LPR without TCP handshake on SUEE8 dependent on strikes (p=0.77687Hz). More strikes result in a higher balanced accuracy but also much higher identification times [5].

best thresholds for the three attacks show that this scheme works very well in detecting each attack (with a balanced accuracy of up to 0.995). However, each of these threshold pairs results in very bad detection rates for the other attacks. When the maxima of the thresholds of each partial scheme are combined, a good balanced accuracy of up to 0.926 without TCP handshake or 0.906 with TCP handshake for all attacks equally can be achieved. The identification times of 13 to 39 seconds are also much better than the ideal thresholds for each attack.

For all these evaluations, one suspicious packet is enough to deem a client an attacker. Meaning, for the SUEE8 data set, a client has to send only one suspicious packet in more than one week. The accuracy can be improved when several strikes, i. e., suspicious packets are necessary for an attacker classification. We conducted all tests with one, two, three, and four strikes. As an example, Figure 47 shows our results for low packet rate without TCP handshake and a fixed threshold of 0.77687Hz for all measurements. The left figure shows that balanced accuracy can be improved when the number of strikes necessary for detection is increased. However, as can be seen in the right figure, this extends the identification time extensively: from a mean of 16 seconds to 61 seconds for slowloris, from 12 seconds to 86 seconds for slowHTTPTest, and from 53 to 67 seconds for slowlorisng. This means a 281 %, 616 % and 26 % increase in identification time. Therefore, the comparably low increase in accuracy does result in impractical identification times for real-time applications. However, there is a clear potential to use strikes for non-time-critical analysis of the data, such as in the case of forensics.

11.7 SUMMARY

In this chapter, we designed, built, and evaluated a mitigation framework against DDoS attacks that works without the victim's help. We

have shown that our framework can detect attacks, identify the attackers, and mitigate the effects of the attack within minutes or even seconds, without or with minimal optimization for the specific network infrastructure or application. The DDoS mitigation framework presented here is a comprehensive framework solving several issues in the area of DDoS mitigation. It combines the detection of attacks, identification of attacking clients, and the mitigation of attacks in one system. Unlike other systems, it mitigates not only network-based attacks but also application-layer attacks — although the system is located and operates within the network infrastructure and its constraints. The system encompasses two different detection mechanisms based on target availability and based on network behavior, and an identification system that can identify attackers for several application-based attacks such as HTTP flood, TLS flood, and several slow attacks. We showed that it is able to successfully mitigate SYN flooding attacks if the network infrastructure itself is strong enough not to be affected by the attack. Moreover, a defense system countering the attacks is part of the framework. For one, identified attackers can be removed from the network or forwarded to a CAPTCHA server. For reflective attacks, it is not feasible to identify the attacking clients due to the heavy load of such attacks. The network infrastructure itself is the target of the attack and cannot be utilized to analyze traffic apart from the bare minimum. Therefore, we introduced a system that can mitigate reflective attacks without identifying the attackers. We designed, implemented, and evaluated a new mitigation mechanism that can reliably detect and mitigate arbitrary DRDoS attacks as long as the underlying protocol uses UDP and a fixed server port. The system can be installed instantly within the network infrastructure — transparent to the attack target. Incoming and outgoing connections to and from the target remain unaffected by the defense mechanism — even if the target uses the very same service that is abused for the attack. The system scales well, and the defense can be spread wide towards the edge of the network infrastructure under control.

While our mechanisms can work with any modern network infrastructure, we used SDN, a comparably new technology, to build our prototype. Although SDN as a technology has been around for several years, the available hardware is still maturing, making tests with a real hardware infrastructure challenging; however, we show that even with such limitations, SDN can be used to mitigate attacks effectively, and the flexibility of SDN can be advantageous going forward.

Slow HTTP attacks differ quite a bit from flooding attacks, and their identification and mitigation can lead to a high management effort of the network infrastructure. We developed several concepts based on light-weight flow-based analysis of network traffic that can identify attackers and help to exclude them from the network. Our analyses

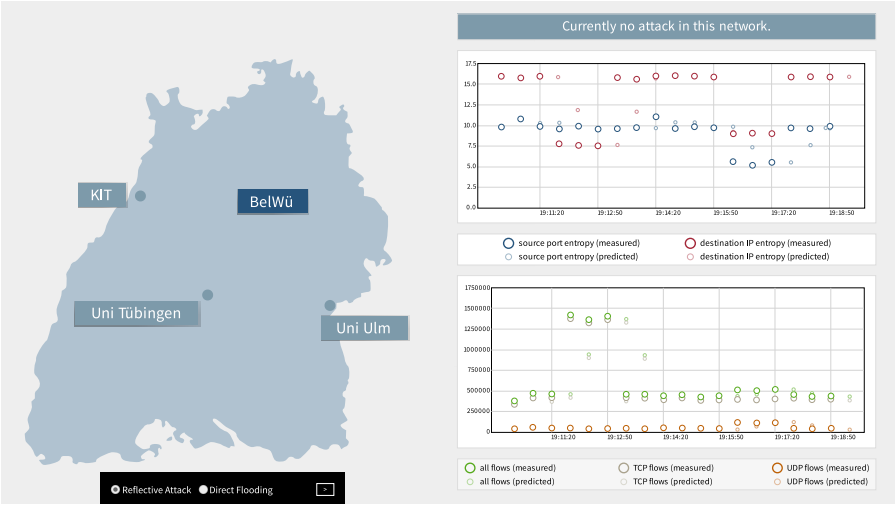


Figure 48: DDoS detection demo running based on live analysis of bwNetFlow data. Additionally to live analyzing the network and finding DDoS attacks, the demo also allows the user to simulate an attack themself.

showed that a network-based defense approach against slow attacks is feasible and should be considered as part of a defense strategy for network providers. The accuracy of the schemes is not high enough to leave the system active all the time, but it is very effective as part of a reactive defense system once ongoing attacks have been identified. The attack tools we used are only able to conduct attacks based on the HTTP protocol. However, as we did not use any scheme that is dependent on application-layer data, our mechanisms should also be able to protect other TCP-based application-layer protocols that are vulnerable to slow attacks.

In this work, we also introduced slowloris-ng, a more sophisticated attack tool compared to slowloris that is harder to detect by network operators due to its more randomized behavior.

11.7.1 *Outlook and Deployment*

As part of the collaboration with the bwNetFlow project, a prototype was built to analyze data in the BelWü network and detect and classify attacks in the network infrastructure. A demo showing the prototype can be seen in Figure 48. This system is planned to be integrated into the Grafana dashboard run by the BelWü and offered as a service to all BelWü clients in the near future.

V

CONCLUSIONS

12

CONCLUSIONS & OUTLOOK

Network security as a field encompasses a wide variety of challenges. There is a multitude of attacks with different goals and different ways of archiving them. There is the aspect of even faster and even bigger networks every year. There are challenges imposed by the ossification of the Internet infrastructure. There are new technologies published regularly that constantly change the face of the Internet and networks in general. In this work, we focused only on a small part of this field: testing security mechanisms, acceleration of IDS, and DDoS mitigation, all in the context of high-bandwidth networks.

The evaluations of network devices and resources are done very differently from research group to research group. Data sets used for evaluations are not proven to reflect general networks and can age rapidly. Data sets for edge cases and the newest technologies are often hard to impossible to obtain.

Intrusion Detection Systems are becoming more and more complex due to more sophisticated attacks, while the amount of network data to analyze rises faster than the computational power. In the past, research groups often looked into hardware-based acceleration of IDS to offload some of the workload to other devices. However, a contemporary analysis to what extend FPGA-based or GPU-based acceleration can help IDS was missing in the literature.

The scale of Denial-of-Service attacks is continuously rising. New types of attacks are emerging regularly. Reliable detection and classification of attacks, identification of the attacking clients, and subsequent defense against the attacks can be tricky. Often, operators of the attack target, network operators of the local network, and Internet service providers have to collaborate to mitigate the attacks.

In this thesis, we looked into these issues of modern networks and found answers to some of the most pressing questions.

Part II
Chapter 2

As groundwork to conduct our evaluations, the resources available for network testing were evaluated. This includes but is not limited to an extensive survey on publicly available data sets.. The data sets were analyzed, described, and classified. The data sets came from simulated networks, event networks, production networks, or a combination of these. They contain either attack traffic, benign traffic, or both. The data sets are partially labeled and represent specific or more generic networks. Moreover, analysis tools and frameworks were described, along with evaluation methodologies. Analyzing the state of the art in network testing led to the following research questions:

Research Question: *Network testing in high-bandwidth networks: Evaluation of TCP congestion control mechanisms in 10G networks as a use case.*

Research Question: *How can benign traffic be modeled and produced to test network mechanisms?*

Research Question: *How can malicious traffic be modeled and produced to test security network mechanisms?*

Chapter 4 As an initial use case for network evaluation in order to gain experience with benchmarking high-bandwidth networks, we filled a gap in research in congestion control algorithm performance in these networks — specifically 10 Gbit/s networks. The analysis of fairness, convergence time, link utilization, and efficiency of the most common loss-based algorithms Reno, Scalable, High-Speed TCP, BIC, CUBIC, and H-TCP showed that adaptations to the network infrastructure have to be made to achieve maximum single flow performance. Furthermore, we were able to give a clear recommendation for CUBIC in general networks and BIC in networks where legacy devices with older TCP variants can be ruled out.

Chapter 5 Following our analysis of data sets and network testing methodology, and based on our experiences in network testing, we built the General Purpose Network Testing Framework (GPNTF). The framework can produce realistic network traffic (both benign and malicious) based on statistical analyses and measurements and can be adapted to future network infrastructure changes.

With GPNTF and several other tools, it was possible to produce data sets that can be used for evaluation where related work left some gaps. We were able to produce specific traffic for web browsing, storage and marketplace, file sharing, and video on demand to allow for specified analysis based on the traffic mix in the network of deployment. Additionally, a trigger data set for Intrusion Detection Systems was created, which can be used for performance tests. The SUEE data sets recorded with the collaboration of the Student Union Electrical Engineering at Ulm University rounds up our data sets list. SUEE1 and SUEE8 are specialized data sets for the evaluation of slow DDoS attacks. Recorded on a web server, they contain realistic web traffic, which was then enriched with labeled attack traffic from three different slow HTTP attack tools.

This work on network testing approaches and the resulting programs and data sets built the foundation for the evaluations of our following contributions.

We analyzed and classified the common approaches to accelerate Intrusion Detection Systems. We can distinguish between mechanisms that try to limit the amount of data they need to analyze (i. e., sampling) and parallelization approaches. Parallelization can be done by using a multitude of conventional systems in parallel (i. e., clustering) or specialized hardware that perfects parallel computing. Hardware that can be used for this purpose can be, for example, FPGAs, ASICs, and GPUs. Literature showed that these mechanisms could be used quite successfully for IDS acceleration. However, an analysis showing whether these systems can still accelerate modern multi-core implementations of IDS was missing. Analyzing the state of the art in hardware-acceleration of IDS led to the following research questions:

Part III
Chapter 6

Research Question: *Based on maintainability, cost, and acceleration potential which of the hardware options is the most promising?*

Research Question: *Based on the decision following the first question, how can this option be implemented?*

Research Question: *Can this option accelerate the IDS sufficiently?*

Chapter 8

We analyzed whether an FPGA-based pre-filter can be used for the acceleration of IDS. The pre-filter performs the — on CPU very resource-intensive task — of regular expression matching in a highly parallelized fashion. However, we soon realized that the space required on the FPGA to built such a system was not available on current hardware, and consequently, we shifted our focus towards GPU-based acceleration.

GPUs can be used to build highly parallelized systems as their processing cores are optimized for that kind of computations. Prior research showed that at least in single-core implementations of IDS, this could lead to a significant advantage over non-accelerated IDS [307]. However, our analysis of both our own implementation based on Snort and of the Suricata implementation showed, that current implementations taking advantage of modern multi-core systems are far more capable than their single-core predecessors and the overhead added particularly by PCIe bus usage and collaboration between CPU and GPU exceeds the performance gain of the GPU. The evaluation was done using the aforementioned General Purpose Testing Framework and its data sets.

We analyzed the state of the art of Distributed Denial-of-Service attacks. We described and classified the most common DDoS attacks. Typical attacks include flooding attacks both on application-layer and lower layers, amplification attacks such as reflective attacks, and slow

Part IV
Chapter 9

attacks. They all have in common that they aim to exhaust one or more resources. However, their strategies are very diverse. This can also be seen when analyzing mitigation mechanisms. Many mechanisms need to be tailored to specific attacks. Analyzing mitigation mechanisms, we found that many only work with the attack target's collaboration, while others impact not only the attackers but also benign traffic in the network. Analyzing the state of the art in DDoS mitigation led to the following research questions:

Research Question: *Under which circumstances do common detection mechanisms detect DDoS attacks reliably?*

Research Question: *Can the common detection mechanisms help to find DDoS attacks in the BelWü network and other research networks with similar characteristics?*

Research Question: *How can the network-based identification of DDoS attacks be improved? Our focus here lies mainly on slow attacks as here the need for improvement has been identified.*

Research Question: *How can the network-based defense against DDoS attacks be improved? We focus here mainly on reflective attacks.*

Chapter 11

We developed detection mechanisms to detect application-layer and network-layer Distributed Denial-of-Service Attacks solely in the network without the cooperation of the attack target. We developed network-based schemes able to identify clients during slow DDoS attacks tested with the aforementioned SUEE data sets. We developed a defense system against reflective DDoS attacks that can be deployed instantly and network-wide.

We build a framework for DDoS mitigation using Software-Defined Networking technology. The framework is able to utilize the mechanisms we developed and was used to implement and test our mechanisms. We extensively analyzed the mechanisms based on this framework, with commonly used data sets (CAIDA, WIDE) and live data from the BelWü network.

A system for DDoS detection and classification for the institutions using the BelWü services was implemented as a demonstrator working on the live data obtained in the BelWü network.

12.1 OUTLOOK

We targeted network testing, acceleration of IDS, and DDoS mitigation in this thesis. In these three areas, we see several points where future work could build upon this work.

Extension of the General Purpose Network Testing Framework

GPNTF is able to produce web traffic, video-on-demand traffic, storage and marketplace traffic, and file sharing traffic. This makes up around 80 % of Internet traffic [267–269]. However, 20 % of Internet traffic cannot be simulated yet. This includes gaming, communications (e. g., Skype, WhatsApp, WeChat), administration (e. g., DNS, NTP, ICMP), and tunneling protocols such as SSH or proxies. Especially administration protocols that do not make up much traffic but are crucial to network operations should be added to the framework.

Intrusion Detection System Acceleration Through ASICs

As part of our research, we analyzed the possibility of accelerating Intrusion Detection Systems by offloading the regular expression matching onto an ASIC. However, due to the immense effort necessary to implement a processor core with a comprehensive feature list that can be used in production, we decided not to pursue this further. Therefore it was not possible for us to conclusively clarify whether this approach could be more promising than the other two approaches.

DDoS Mitigation Framework Distribution

The framework implemented for our detection, identification, and defense mechanisms reliably worked as a base for our evaluations. However, the identification and defense aspects are only implemented on one machine per task, meaning one machine running the SDN controller, one machine running the network analyzer, et cetera. We undertook the first steps to distribute our system further. The concept we have in mind consists of several autonomous systems spread in the network. Each system contains one SDN controller, analyzer, and observer responsible for defending a known list of potential targets and controlling the bottleneck link to these targets. The SDN controllers of each system are aware of each other and exchange attack information with each other. One central server acts solely as a certificate authority when a new SDN controller is added to the network. Apart from that, the SDN controllers communicate through a publish-subscribe pattern. A first prototype of this was implemented as a proof-of-concept. We suspect that such a distributed system could react faster and more reliably to attacks. A final investigation of whether this is true or not is still pending.

New Network Technologies and Their Impact

In the last couple of years, many new protocols and mechanisms in networks have found their way into production networks. QUIC

slowly becomes the standard transport layer protocol and might fully replace TCP at some point. Simultaneously, Bottleneck Bandwidth and Round-trip propagation time (BBR) as a TCP and QUIC congestion control algorithm further change network behavior. These protocols could affect our DDoS mitigation or even allow new attacks to be implemented that cannot be detected as of now. Analysis of these and other protocols for potential exploitation in the context of Denial-of-Service attacks and developing mitigation mechanisms could prove beneficial.

DDoS Mitigation Deployment in BelWü

The first DDoS detection prototype based on the bwNetFlow data was already implemented as a proof-of-concept demonstrator. The next step will be to integrate it into the Grafana Dashboard bwNetFlow develops and maintains. The dashboard can be used by all clients of the BelWü network; the data analysis limited in scope to the subnet of this specific customer. Detection and classification of DDoS attacks could help network administrators to react to security incidents of this kind faster and more accurately.

DDoS Mitigation in Self-Driving Networks

Self-driving networks is a marketing term coined by Juniper[1] that encompasses the idea of autonomous network configuration. These networks shall be predictive of changes and shall adapt automatically based on observations made in the network. In the context of security, a DDoS mitigation framework that reacts automatically to a Denial-of-Service attack and changes the network configuration can be seen as a self-driving network component. Integrating our system in such an environment should be possible and should be researched. The opportunities of a self-driving network could further advance the mitigation of DDoS attacks. While the current system only reacts to attacks by reconfiguration of routers and switches to block traffic or change IP addresses of flows from the target, in a self-driving network, a complete autonomous reconfiguration of the network would be possible.

12.2 SUMMARY

With this thesis, we tackled several areas of high-bandwidth network security research. We improved network testing, gave new insights into the applicability of hardware-acceleration of IDS, and improved Distributed Denial-of-Service attack mitigation. We published a new

1 https://www.juniper.net/us/en/insights/the-self-driving-network/

testing framework (GPNTF), published several data sets assisting future research, and supported BelWü providing a new DDoS detection system.

ACKNOWLEDGEMENTS

Many have influenced me, helped me, supported me, put up with me when I was in a bad mood, and always pushed me forward. Without them, this thesis would not have been possible.

I would like to thank Prof. Dr. rer. nat. Frank Kargl, for his guidance and supervision over the years, for giving me a lot of academic freedom, and for proofreading this thesis. I thank prof. dr. ir. Aiko Pras for offering to act as the second, external examiner of this thesis and Prof. Dr.-Ing. Dr. h.c. Stefan Wesner for offering to act as the third examiner.

Further, I would like to thank my colleagues in the bwNET100G+ and bwNetFlow projects for the great work over the years and the state of Baden-Württemberg for giving us the opportunity to work in these projects.

During my time at the Institute of Distributed Systems, I was lucky to be able to supervise many bright and dedicated students. It was a blast working with you, thank you.

I would also like to thank my current and former colleagues for many productive discussions, their help, and their inspiring work ethic; but also for the coffee breaks, Cafeteria tours, and the fun we had at PhD defenses, institute retreats, and Christmas parties. I am lucky that I can call many of you my friends. Special thanks go to Benjamin Erb and Leonard Bradatsch for proofreading parts of this thesis.

Danke to my parents Barbara and Mario for being there for me, supporting me over the years and decades through school and university, pushing me, guiding me, and giving me the feeling that I can always depend on them.

Thank you to all my friends, and a special *thank you*, *bedankt*, *danke*, *xièxiè*, and *terima kasih* goes to the *Tafelrunde*. Thank you for living and traveling with me — for exploring the wonders of this world with me. In particular, *danke* and *xièxiè* Leo and Surong for taking me with you to China and *terima kasih* Clara for showing me Indonesia. Thank you for many nights with pancakes, hot pot, music, movies, wine, beer, and games — both the analog and the digital kind. *Thank you* Katja for many afternoons with coffee and Laugencroissants and *danke* Silke for many lunches all around Ulm. You helped me stay sane during the time of writing. Thank you, *IomG*, for the games.

Last but certainly not least, *terima kasih* to the lovely Clara Trias Winda Rahajeng for always being there for me. Thank you for the incredible food, company, cute cat GIFs, and distraction when I needed it most.

VI

APPENDIX

Table 16: GPNTF default values for web traffic and their sources.

	Measured / Literature Values	**Model Default**
request size	mean: 318.59 B σ = 197.46 B [98]	318 B
# of main objects	Mean: 2.19 Median: 1 Max: 212 σ = 2.63 [252]	1
main object size	Mean: 31.5 kB Median: 19.4 kB Max: 8 MB σ = 49.2 kB [252]	Weibull shape = 0.814944 scale = 28242.8
parsing time	-	0
# of inline objects	Mean: 31.93 Median: 22 Max: 1920 σ = 37.65 [252]	Exponential $\lambda = 3.132 \cdot 10^{-6}$
inline object size	Mean: 23.9 kB Median: 10.2 kB Max: 8 MB σ = 128 kB [252]	Lognormal μ = 9.17979 σ = 1.24646
# of TCP connections	6 (Firefox 59.1, measured)	1
pipelining / multiplexing	-	No/No
# of servers	Min: 1, Max: 70 [78]	1
HTTP compression	-	No
caching	-	No
reading time	Mean: 39.7 s Max: 10.000s σ = 325 s [98]	Lognormal μ = -0.495204 σ = 2.7731

Table 17: GPNTF default values for file sharing and their sources.

	Measured / Literature Values	**Model Default**
session length	-	10
file size	-	-
chunk size	-	-
flow size	mean: 362.4 kB median: 1.17 kB σ = 12470 B [58]	concatenation of bounded Weibull and Pareto distribution
packet size	~62% big packets ($\geqslant$ 1350 Bytes) ~38% small packets ($\leqslant$ 67 Bytes)[185]	~62% big packets (1500 Bytes) ~38% small packets (62 Bytes)
# of concurrent download connections	max. 270 recommended[314]	defined by user
# of concurrent flows per IP	~1[58]	1
bandwidth limit	-	-
idle time	-	0 s
hit rate	-	-
tit-for-tat	-	-
# of allowed concurrent upload connections	max. 90 recommended[314]	90
freerider	-	-

Table 18: GPNTF default values for buffered video streaming and their sources.

	Measured / Literature Values	**Model Default**
streaming technique	-	MPEG-DASH
video length	20 min, 30 min, 40 min[183]	~10 min to ~90 min available, user defined
video quality	1% UHD, 76% HD, 23% SD[99]	240p to 1080p available, user defined
HTTP version	-	HTTP/1.1
manifest	-	MPD
segment length	1 s to 4 s[256]	1 s to 15 s available, user defined
adaptive streaming algorithm	-	VLC default
requesting rate	roughly corresponding to segment length[256]	VLC default
segment download approach	-	VLC default

Table 19: GPNTF default values for storage and marketplace and their sources. (Own measurements if no source is cited).

	Measured / Literature Values	**Model Default**
file size	popular file sizes between 100 KB and 20 GB (tracked 2018-08-10 to 2018-08-16 on demonoid.pw)	878 MB
packet size	data packets: MTU size control packets: < 101 Bytes[173]	data packets: MTU size
# of connections	one connection per download	one connection
layer 7 protocol	FTP RFC172, HTTP/1.1	-
idle time	-	0 s

Theorem 1. $A \setminus B \setminus C = A \setminus C \setminus B$

Proof of Theorem 1. Properties needed for proof:

1. $A \cup B = B \cup A$ (Commutativity)
2. $\overline{A \setminus B} = \overline{A} \cup B$ (Relationship between absolute and relative complement)

$$\begin{aligned} A \setminus B \setminus C &\overset{2}{=} \overline{(\overline{A} \cup B)} \setminus C \\ &\overset{2}{=} \overline{((\overline{A} \cup B) \cup C)} \\ &\overset{1}{=} \overline{((\overline{A} \cup C) \cup B)} \\ &\overset{2}{=} \overline{((\overline{A} \cup C)} \setminus B) \\ &\overset{2}{=} A \setminus C \setminus B \end{aligned} \tag{21}$$

□

Theorem 2. $\overline{A} \setminus \overline{B} = B \setminus A$

Proof of Theorem 2. Properties needed for proof:

3. $A \setminus B \setminus C = A \setminus C \setminus B$ (Theorem 1)
4. $\overline{A} = U \setminus A$ with the universal set U
5. $(A \cup B) \setminus C = (A \setminus C) \cap (B \setminus C)$ (Distributivity)

$$\begin{aligned} \overline{A} \setminus \overline{B} &\overset{4}{=} (U \setminus A) \setminus \overline{B} \\ &= ((B \cup \overline{B}) \setminus A) \setminus \overline{B} \\ &\overset{5}{=} ((B \setminus A) \cup (\overline{B} \setminus A)) \setminus \overline{B} \\ &\overset{5}{=} ((B \setminus A) \setminus \overline{B}) \cup ((\overline{B} \setminus A) \setminus \overline{B}) \\ &\overset{3}{=} ((B \setminus \overline{B}) \setminus A) \cup ((\overline{B} \setminus \overline{B}) \setminus A) \\ &= B \setminus A \end{aligned} \tag{22}$$

□

Table 20: Full list of all fields contained in the bwNetFlow data (part 1). Partially based on the official GoFlow documentation [101]; some values supplemented by bwNetFlow.

Label	Description
Type	Type of flow message
TimeReceived	Timestamp of when the message was received
SequenceNum	Sequence number of the flow packet
SamplingRate	Sampling rate of the flow
FlowDirection	Direction of the flow
SamplerAddress	Address of the device that generated the packet
TimeFlowStart	Time the flow started
TimeFlowEnd	Time the flow ended
Bytes	Number of bytes in flow
Packets	Number of packets in flow
SrcAddr	Source address (IP)
DstAddr	Destination address (IP)
Etype	Ethernet type (0x86dd for IPv6...)
Proto	Protocol numbers (e. g., 17=UDP, 6=TCP, 1=ICMP...)
SrcPort	Source port (when UDP/TCP/SCTP)
DstPort	Destination port (when UDP/TCP/SCTP)
SrcIf	Source interface
DstIf	Destination interface
SrcMac	Source mac address
DstMac	Destination mac address
SrcVlan	Source VLAN ID
DstVlan	Destination VLAN ID
VlanId	802.11q VLAN ID
IngressVrfID	VRF ID
EgressVrfID	VRF ID
IPTos	IP Type of Service
ForwardingStatus	Forwarding status

Table 21: Full list of all fields contained in the bwNetFlow data (part 2). Partially based on the official GoFlow documentation [101]; some values supplemented by bwNetFlow.

Label	**Description**
IPTTL	IP Time to Live
TCPFlags	TCP flags
IcmpType	ICMP Type
IcmpCode	ICMP Code
IPv6FlowLabel	IPv6 Flow Label
IPv6ExtensionHeaders	IPv6 Extension Headers
FragmentId	IP Fragment ID
FragmentOffset	IP Fragment Offset
BiFlowDirection	BiFlow Identification
SrcAS	Source AS number
DstAS	Destination AS number
NextHop	Nexthop address
NextHopAS	Nexthop AS number
SrcNet	Source address mask
DstNet	Destination address mask
Cid	Internal customer number in the BelWü network
CidString	Internal customer in the BelWü network
Normalized	Binary value (0 or 1)
SrcIfName	Source interface name
SrcIfDesc	Source interface description
SrcIfSpeed	Speed at interface in Mbit
DstIfName	Destination interface name
DstIfDesc	Source interface description
DstIfSpeed	Speed at interface in Mbit
ProtoName	Protocol (UDP, TCP, ICMP...)
RemoteCountry	ISO 3166-1 alpha-2 country code

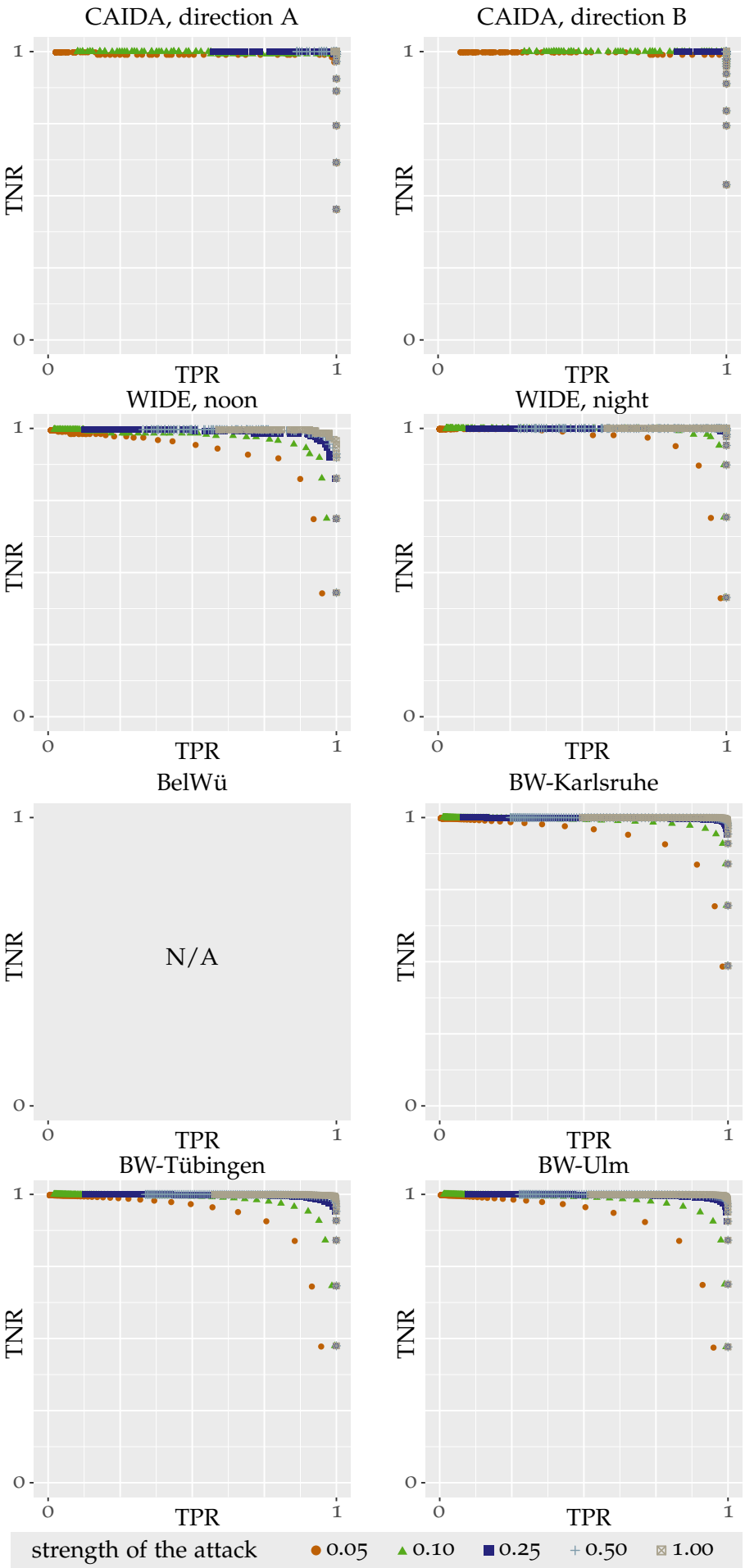


Figure 49: ROC curves for the destination IP entropy (analyzing every 128th packet)

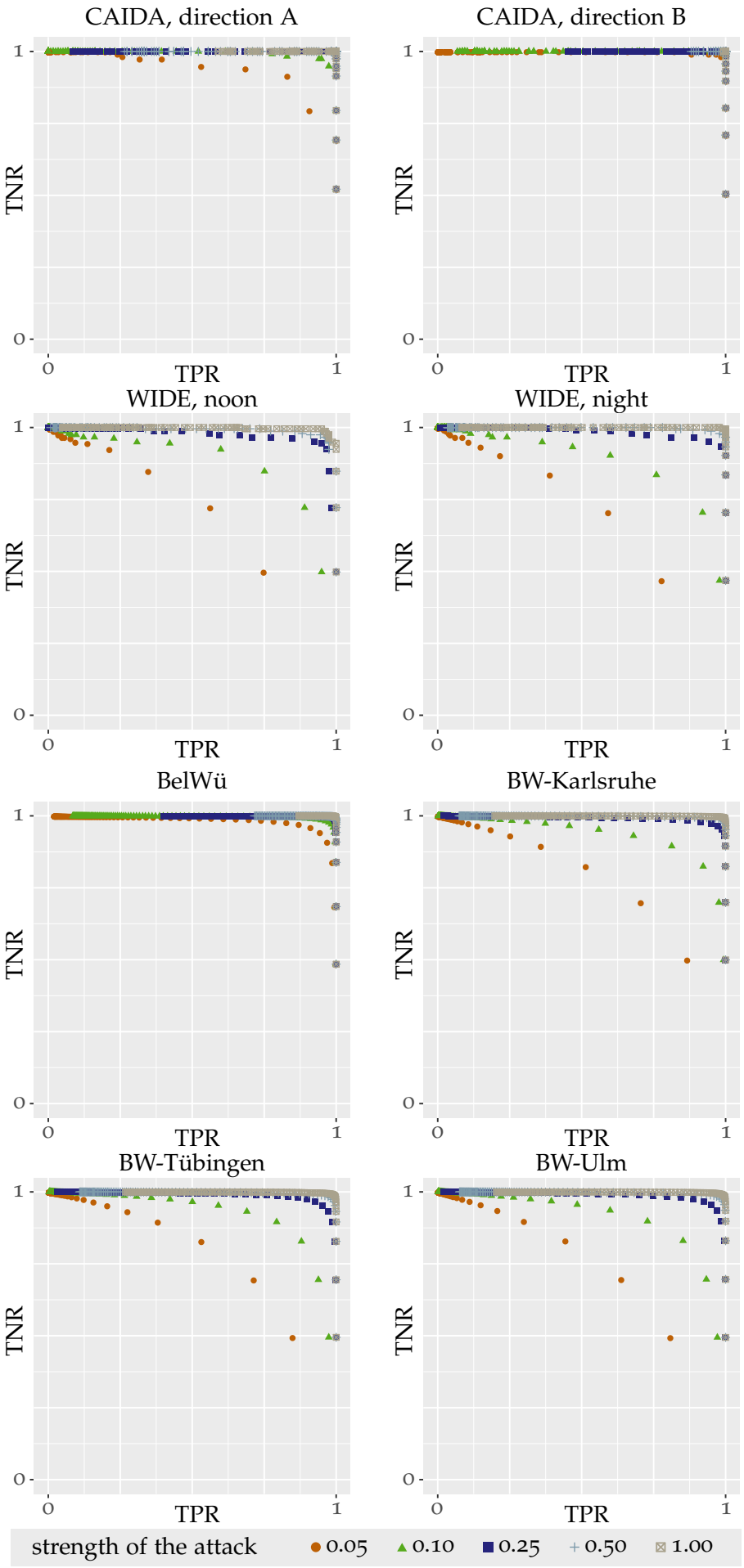


Figure 50: ROC curves for the destination IP entropy (analyzing every 2 048th packet)

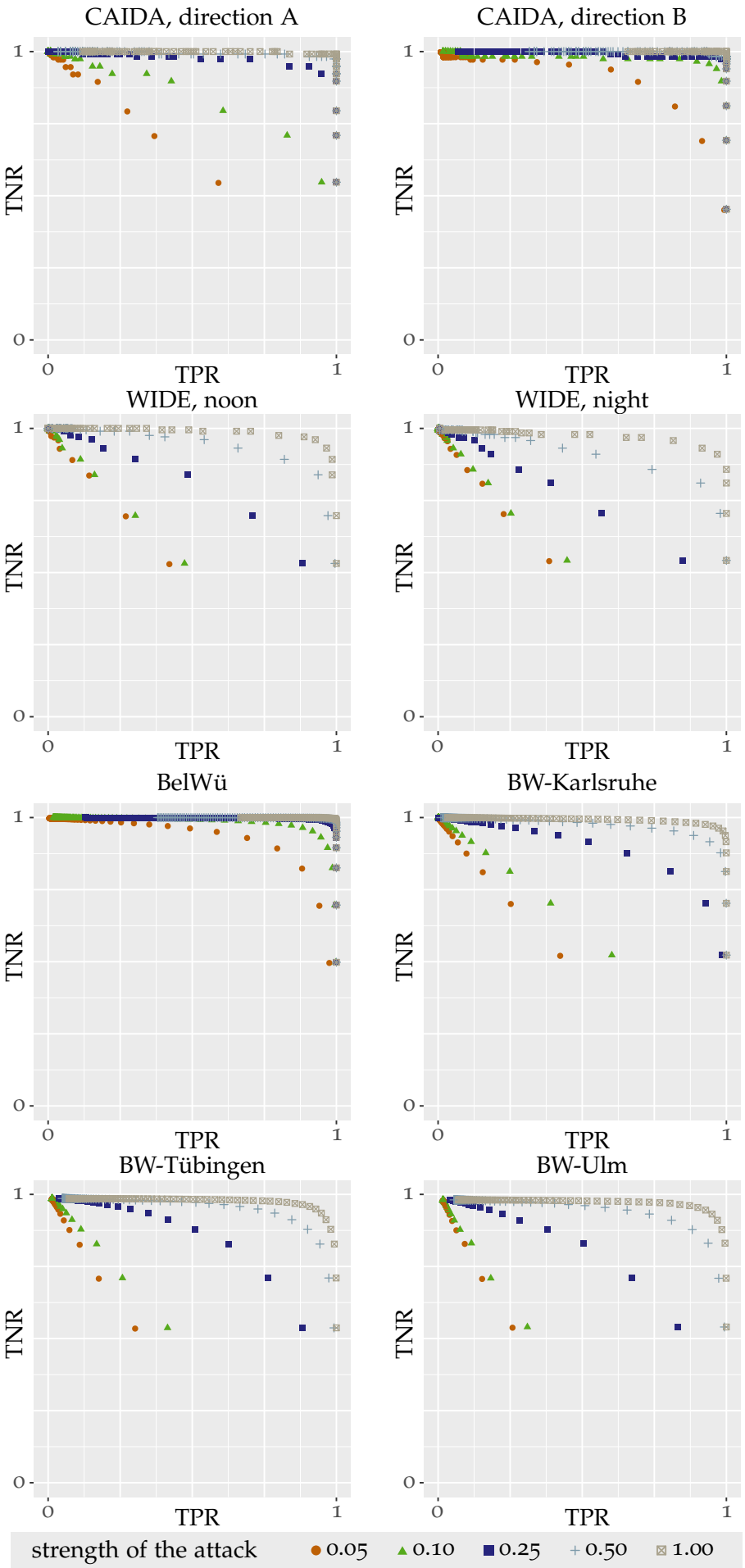


Figure 51: ROC curves for the destination IP entropy (analyzing every 32 768th packet)

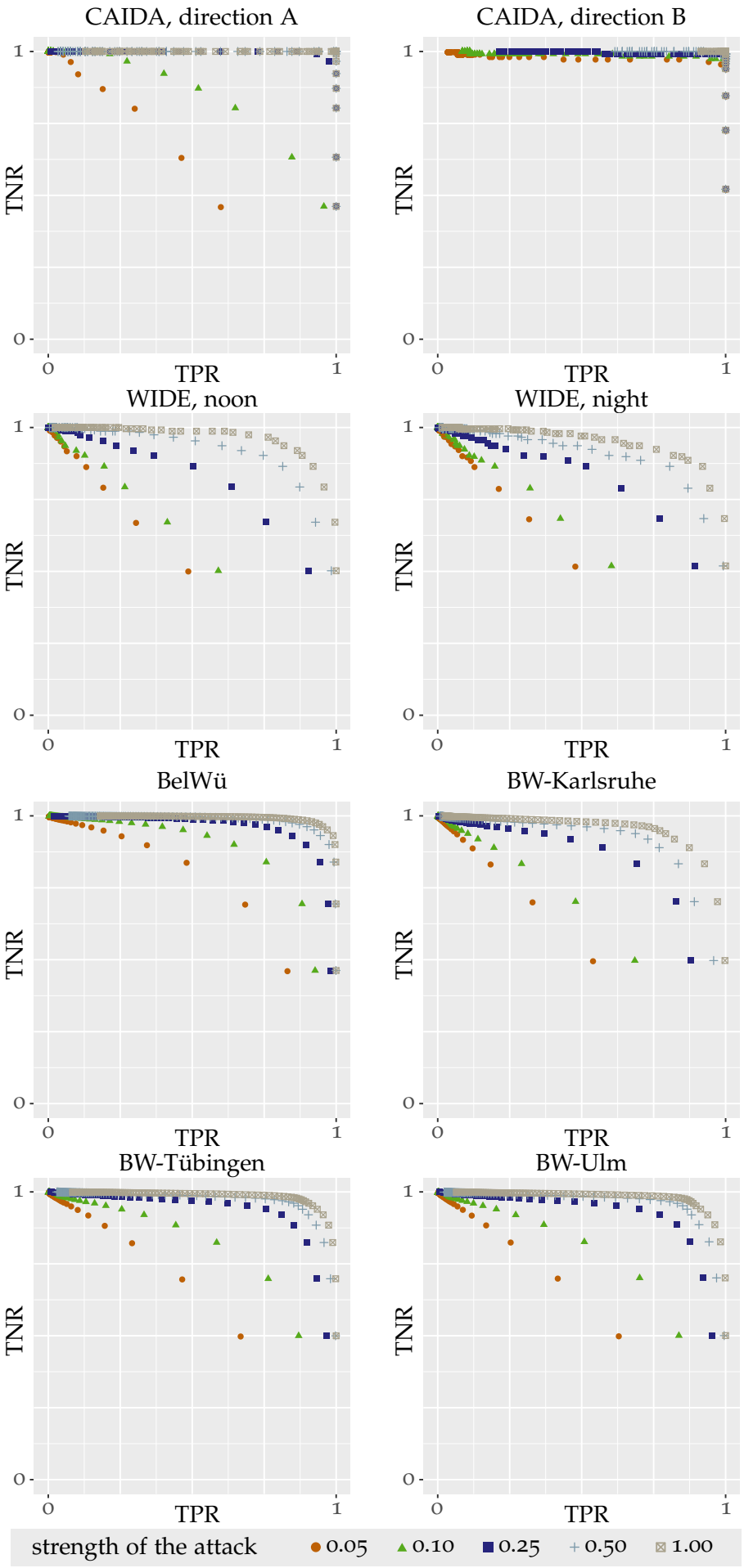


Figure 52: ROC curves for the source port entropy (analyzing every 128th packet)

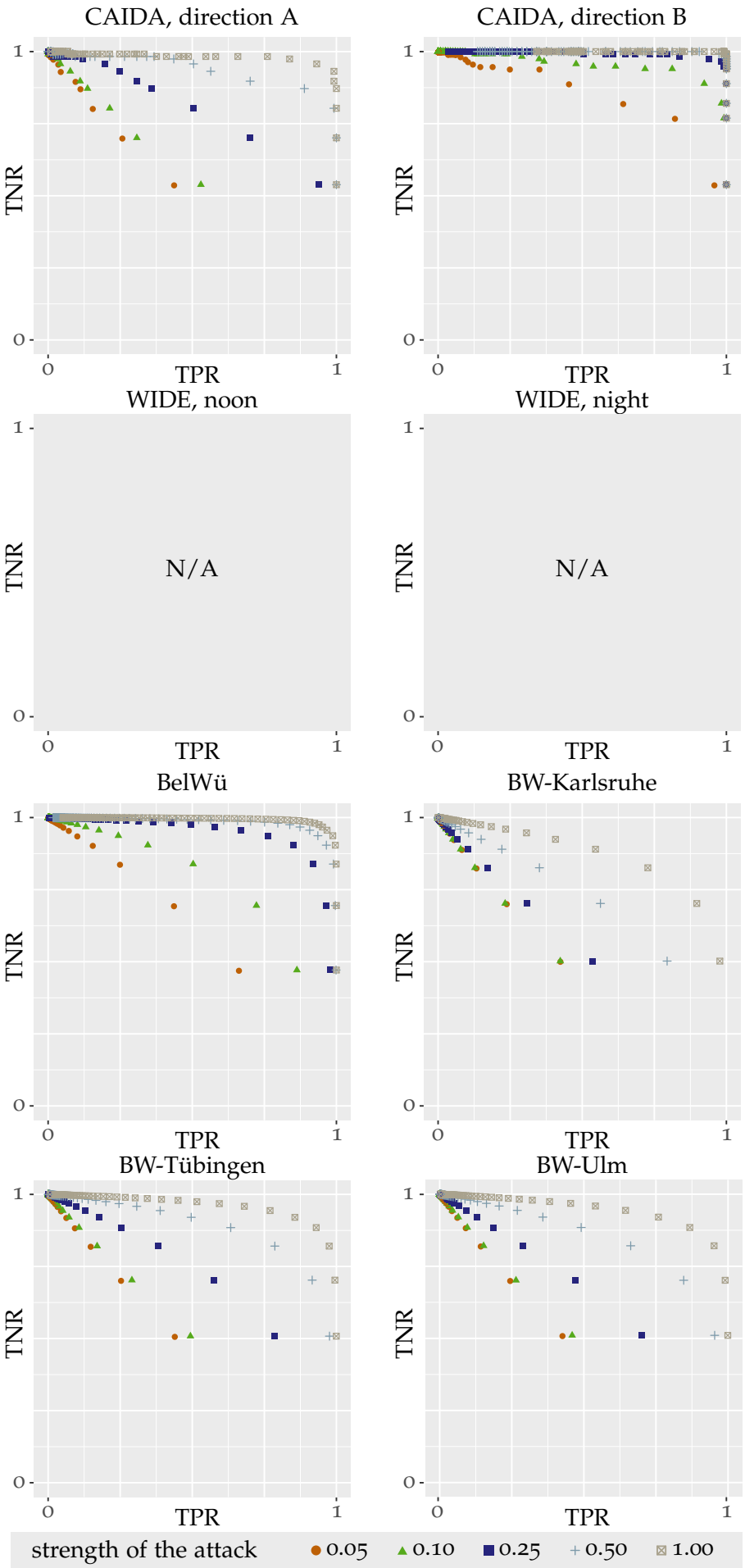


Figure 53: ROC curves for the source port entropy (analyzing every 2 048th packet)

BIBLIOGRAPHY

AUTHOR'S PUBLICATIONS RELEVANT TO THIS DISSERTATION

[1] **T. Lukaseder**, L. Bradatsch, B. Erb, R. W. van der Heijden, and F. Kargl. "A Comparison of TCP Congestion Control Algorithms in 10G Networks." In: *IEEE 41st Conference on Local Computer Networks (LCN)*. Oct. 2016. DOI: 10.1109/LCN.2016.121 (cit. on pp. 7, 37, 43, 55, 57, 59, 60, 62, 63).

[2] **T. Lukaseder**, L. Bradatsch, B. Erb, and F. Kargl. "Setting Up a TCP Benchmarking Environment—Lessons Learned." In: *IEEE 41st Conference on Local Computer Networks (LCN)*. Oct. 2016. DOI: 10.1109/LCN.2016.32 (cit. on pp. 7, 37, 39, 40, 43).

[3] **T. Lukaseder**, J. Fiedler, and F. Kargl. "Performance Evaluation in High-Speed Networks by the Example of Intrusion Detection Systems." In: *11. DFN-Forum Kommunikationstechnologien*. 2018 (cit. on p. 8).

[4] **T. Lukaseder**, A. Hunt, C. Stehle, D. Wagner, R. van der Heijden, and F. Kargl. "An Extensible Host-Agnostic Framework for SDN-Assisted DDoS-Mitigation." In: *IEEE 42nd Conference on Local Computer Networks (LCN)*. Oct. 2017. DOI: 10.1109/LCN.2017.103 (cit. on pp. 8, 158, 178, 181).

[5] **T. Lukaseder**, L. Maile, B. Erb, and F. Kargl. "SDN-Assisted Network-Based Mitigation of Slow DDoS Attacks." In: *EAI International Conference on Security and Privacy in Communication Networks (SecureComm)*. Aug. 2018. DOI: 10.1007/978-3-030-01704-0_6 (cit. on pp. 9, 79, 166, 198, 199).

[6] **T. Lukaseder**, L. Maile, and F. Kargl. "SDN-Assisted Network-Based Mitigation of Slow HTTP Attacks." In: *1. KuVS Fachgespräch Network Softwarization – From Research to Application*. 2017. DOI: 10.15496/publikation-19543 (cit. on pp. 9, 166).

[7] **T. Lukaseder**, K. Stölzle, S. Kleber, B. Erb, and F. Kargl. "An SDN-based Approach For Defending Against Reflective DDoS Attacks." In: *IEEE 43rd Conference on Local Computer Networks (LCN)*. Oct. 2018. DOI: 10.1109/LCN.2018.8638036 (cit. on pp. 9, 170).

[8] **T. Lukaseder**, S. Ghosh, and F. Kargl. "Mitigation of Flooding and Slow DDoS Attacks in a Software-Defined Network." In: *IEEE 43rd Conference on Local Computer Networks (LCN), Demo Track*. Oct. 2018 (cit. on p. 8).

[9] L. Bradatsch, **T. Lukaseder**, and F. Kargl. "A Testing Framework for High-Speed Network and Security Devices." In: *IEEE 42nd Conference on Local Computer Networks (LCN)*. Oct. 2017. DOI: 10.1109/LCN.2017.91 (cit. on pp. 7, 65).

[10] F. Engelmann, **T. Lukaseder**, B. Erb, R. van der Heijden, and F. Kargl. "Dynamic packet-filtering in high-speed networks using NetFPGAs." In: *IEEE Third International Conference on Future Generation Communication Technologies (FGCT 2014)*. Aug. 2014. DOI: 10.1109/FGCT.2014.6933224 (cit. on pp. 8, 110).

OTHER PUBLICATIONS BY THE AUTHOR

[11] R. van der Heijden, **T. Lukaseder**, and F. Kargl. "VeReMi: A Dataset for Comparable Evaluation of Misbehavior Detection in VANETs." In: *EAI International Conference on Security and Privacy in Communication Networks (SecureComm)*. 2018.

[12] **T. Lukaseder**, M. Halter, and F. Kargl. "Context-based Access Control and Trust Scores in Zero Trust Campus Networks." In: *GI Sicherheit*. (accepted). 2020. DOI: 10.18420/sicherheit2020_04.

[13] R. van der Heijden, **T. Lukaseder**, and F. Kargl. "Analyzing attacks on cooperative adaptive cruise control (CACC)." In: *2017 IEEE Vehicular Networking Conference (VNC)*. Nov. 2017, pp. 45–52. DOI: 10.1109/VNC.2017.8275598.

[14] C. Corbett, T. Basic, **T. Lukaseder**, and F. Kargl. "A Testing Framework Architecture for Automotive Intrusion Detection Systems." In: *Automotive - Safety & Security 2017 - Sicherheit und Zuverlässigkeit für automobile Informationstechnik*. Ed. by P. Dencker, H. Klenk, H. B. Keller, and E. Plödererder. Gesellschaft für Informatik, Bonn, 2017, pp. 89–102.

CO-SUPERVISED STUDENT THESES, PROJECTS, AND INTERNSHIPS (UNPUBLISHED)

[15] L. Bradatsch. *Verhalten von TCP in Hochgeschwindigkeitsnetzen*. Bachelor Thesis, VS-Bo8-2015. Sept. 2015 (cit. on pp. 37, 43).

[16] L. Bradatsch. *General Purpose Network Testing Framework (GPNTF)*. Master Project. Aug. 2016 (cit. on p. 65).

[17] L. Bradatsch. *Determination of Traffic Models for Network Testing*. Master Thesis, VS-M10-2018. Aug. 2018 (cit. on pp. 29, 65).

[18] L. Elzobaidy. *SDN Assisted Distributed DDoS Attack Mitigation*. Bachelor Thesis, VS-B15-2017. Aug. 2017.

[19] J. Fiedler. *Performance Evaluation of Intrusion Detection Systems*. Master Project. Apr. 2017.

[20] C. Forst. *Erstellung eines dynamischen Testdatensets zur Sicherheitsanalyse*. Master Thesis, VS-M11-2016. July 2016 (cit. on pp. 16, 33).

[21] S. Ghosh. *Adaptation of an SDN-based DDoS mitigation system from a hardware deployment to a virtualized environment based on Mininet*. Internship. Aug. 2018.

[22] S. Ghosh. *Implementation of a two factor authentication mechanism in form of a mobile app for android and a web site*. Internship. Feb. 2019.

[23] M. Halter. *Anwendung des Zero Trust Modells auf das Forschungsnetz Baden-Württembergs*. Master Thesis, VS-M12-2019. May 2019.

[24] A. Hunt. *Erkennung und Abwehr von Denial of Service Attacken mit Hilfe von Software Defined Networking*. Bachelor Thesis, VS-B06-2016. Mar. 2016 (cit. on p. 158).

[25] A. Hunt, D. Wagner, and C. Stehle. *SDN-Assisted DoS-Mitigation*. Master Project. June 2016 (cit. on p. 158).

[26] D. Lang. *Accelerating Network Intrusion Detection Using a NetFPGA Pre-Filter*. Master Thesis, VS-M14-2015. Sept. 2015 (cit. on pp. 104, 111, 112).

[27] L. Maile. *Extending the SDN-Assisted DDoS-Mitigation Framework*. Master Project. Oct. 2017 (cit. on p. 166).

[28] M. Mohr. *DDoS Detection Based on Traffic Analysis*. Bachelor Thesis, VS-B12-2018. Nov. 2018.

[29] T. Nieß. *DoS-Attacks on Coexistence-Mechanisms for TCP-Variants*. Bachelor Thesis, VS-B11-2017. May 2017.

[30] T. Nieß. *DoS-Attack and Mitigation on TCP Congestion Control Algorithm Coexistence*. Master Project. Sept. 2018.

[31] B. Schimmele. *Implementation of a Pre-Filter for Network Intrusion Detection Systems*. Diploma Thesis, VS-D12-2016. Sept. 2016 (cit. on p. 111).

[32] K. Shymbarova. *Entwicklung einer RegEx Engine für den FPGA-Einsatz*. Master Project. Oct. 2016 (cit. on p. 117).

[33] P. Spiegelt. *Extension and Evaluation of the General Purpose Network Testing Framework*. Bachelor Thesis, VS-B11-2019. June 2019 (cit. on pp. 71, 75–78).

[34] C. Stehle. *Merging and Evaluating Frameworks for SDN-Assisted DDoS-Mitigation*. Master Thesis, VS-M13-2018. Dec. 2018 (cit. on pp. 178, 179).

[35] K. Stölzle. *Defending Against DRDoS Attacks in a High-Speed Network Using an SDN-based Approach*. Master Thesis, VS-M14-2017. Oct. 2017 (cit. on p. 170).

[36] M. Strobel. *Untersuchung der Sicherheit von Eduroam (IEEE802.1X)*. Bachelor Thesis, VS-B15-2015. Dec. 2015.

[37] S. Tomm. *Analyse der Sicherheitsaspekte von VoIP*. Bachelor Thesis, VS-B16-2015. Dec. 2015.

[38] M. Wagner. *GPU-assisted IDS Acceleration*. Bachelor Thesis, VS-B11-2018. Nov. 2018 (cit. on pp. 119, 121, 122).

[39] M. Wagner. *Extending the GPU-Assisted IDS Evaluation*. Master Project. Sept. 2019 (cit. on pp. 80, 119, 124, 125).

[40] J. Ziegler. *Entwicklung eines Modells zur Generierung von Testnetzwerken*. Bachelor Thesis, VS-B20-2018. Nov. 2018.

REFERENCES

[41] M. Abu Rajab, J. Zarfoss, F. Monrose, and A. Terzis. "A Multifaceted Approach to Understanding the Botnet Phenomenon." In: *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*. IMC '06. Rio de Janeriro, Brazil: ACM, 2006, pp. 41–52. ISBN: 1-59593-561-4. DOI: 10.1145/1177080.1177086. URL: http://doi.acm.org/10.1145/1177080.1177086 (cit. on p. 132).

[42] S. Afridi, A. Gilal, S. Shah, and M. Sandhu. "Peer and File Sizes Distributions for Energy Efficient Bit Torrent Networks." In: *IJCSNS International Journal of Computer Science and Network Security* 17.11 (2017) (cit. on pp. 30, 68).

[43] M. E. Ahmed, H. Kim, and M. Park. "Mitigating DNS query-based DDoS attacks with machine learning on software-defined networking." In: *MILCOM 2017 - 2017 IEEE Military Communications Conference (MILCOM)*. Oct. 2017, pp. 11–16. DOI: 10.1109/MILCOM.2017.8170802 (cit. on pp. 142, 145, 152).

[44] L. von Ahn, M. Blum, N. J. Hopper, and J. Langford. "CAPTCHA: Using Hard AI Problems for Security." In: *Advances in Cryptology — EUROCRYPT 2003*. Ed. by E. Biham. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 294–311. ISBN: 978-3-540-39200-2 (cit. on p. 148).

[45] A. V. Aho and M. J. Corasick. "Efficient String Matching: An Aid to Bibliographic Search." In: *Commun. ACM* 18.6 (June 1975), pp. 333–340. ISSN: 0001-0782. DOI: 10.1145/360825.360855. URL: http://doi.acm.org/10.1145/360825.360855 (cit. on pp. 88, 89).

[46] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools (2Nd Edition)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2006 (cit. on p. 105).

[47] A. A. Aizuddin, M. Atan, M. Norulazmi, M. M. Noor, S. Akimi, and Z. Abidin. "DNS Amplification Attack Detection and Mitigation via sFlow with Security-centric SDN." In: *Proceedings of the 11th International Conference on Ubiquitous Information Management and Communication*. IMCOM '17. Beppu, Japan: ACM, 2017, 3:1–3:7. ISBN: 978-1-4503-4888-1. DOI: 10.1145/3022227.3022230. URL: http://doi.acm.org/10.1145/3022227.3022230 (cit. on pp. 142, 152).

[48] A. Alenazi, I. Traore, K. Ganame, and I. Woungang. "Holistic Model for HTTP Botnet Detection Based on DNS Traffic Analysis." In: *Intelligent, Secure, and Dependable Systems in Distributed and Cloud Environments. ISDDC 2017*. 2017 (cit. on p. 27).

[49] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. "Data Center TCP (DCTCP)." In: *Proceedings of the ACM SIGCOMM 2010 Conference*. SIGCOMM '10. New Delhi, India: ACM, 2010, pp. 63–74. DOI: 10.1145/1851182.1851192 (cit. on p. 52).

[50] A. Alshamrani, A. Chowdhary, S. Pisharody, D. Lu, and D. Huang. "A Defense System for Defeating DDoS Attacks in SDN Based Networks." In: *Proceedings of the 15th ACM International Symposium on Mobility Management and Wireless Access*. MobiWac '17. Miami, Florida, USA: ACM, 2017, pp. 83–92. ISBN: 978-1-4503-5163-8. DOI: 10.1145/3132062.3132074. URL: http://doi.acm.org/10.1145/3132062.3132074 (cit. on pp. 142, 145, 152).

[51] J. Amann, S. Hall, and R. Sommer. "Count Me In: Viable Distributed Summary Statistics for Securing High-Speed Networks." In: *Research in Attacks, Intrusions and Defenses*. Ed. by A. Stavrou, H. Bos, and G. Portokalidis. Lecture Notes in Computer Science 8688. Springer International Publishing, 2014, pp. 320–340. ISBN: 978-3-319-11378-4, 978-3-319-11379-1. URL: http://link.springer.com/chapter/10.1007/978-3-319-11379-1%5C_16 (cit. on p. 96).

[52] S. Antonatos, K. G. Anagnostakis, and E. P. Markatos. "Generating Realistic Workloads for Network Intrusion Detection Systems." In: *Proceedings of the 4th International Workshop on Software and Performance*. WOSP '04. Redwood Shores, California: ACM, 2004, pp. 207–215. ISBN: 1-58113-673-0. DOI: 10.1145/974044.974078. URL: http://doi.acm.org/10.1145/974044.974078 (cit. on p. 99).

[53] F. Audet and C. Jennings. *Network Address Translation (NAT) Behavioral Requirements for Unicast UDP*. RFC 4787. http://www.rfc-editor.org/rfc/rfc4787.txt. Jan. 2007 (cit. on p. 173).

[54] S. Axelsson. "Intrusion Detection Systems : A Survey and Taxonomy." In: *Computer Engineering*. 2000, pp. 1–27. DOI: 10.1.1.1.6603. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.83.3043&rep=rep1&type=pdf (cit. on p. 87).

[55] P. E. Ayres, H. Sun, H. J. Chao, and W. C. Lau. "ALPi: A DDoS Defense System for High-Speed Networks." In: *IEEE Journal on Selected Areas in Communications* 24.10 (Oct. 2006), pp. 1864–1876. ISSN: 0733-8716. DOI: 10.1109/JSAC.2006.877136 (cit. on p. 146).

[56] F. Baker and P. Savola. *Ingress Filtering for Multihomed Networks*. RFC 3704. http://www.rfc-editor.org/rfc/rfc3704.txt. Mar. 2004 (cit. on p. 146).

[57] Z. K. Baker, H. j. Jung, and V. K. Prasanna. "Regular Expression Software Deceleration for Intrusion Detection Systems." In: *International Conference on Field Programmable Logic and Applications*. 2006. DOI: 10.1109/FPL.2006.311246 (cit. on p. 97).

[58] N. Basher, A. Mahanti, A. Mahanti, C. Williamson, and M. Arlitt. "A Comparative Analysis of Web and Peer-to-peer Traffic." In: *Proceedings of the 17th International Conference on World Wide Web*. WWW '08. Beijing, China: ACM, 2008, pp. 287–296. ISBN: 978-1-60558-085-2. DOI: 10.1145/1367497.1367537. URL: http://doi.acm.org/10.1145/1367497.1367537 (cit. on pp. 30, 68, 218).

[59] F. Beer, T. Hofer, D. Karimi, and U. Bühler. "A new Attack Composition for Network Security." In: *10. DFN-Forum Kommunikationstechnologien*. Ed. by P. Müller, B. Neumair, H. Raiser, and G. Dreo Rodosek. Bonn: Gesellschaft für Informatik e.V., 2017, pp. 11–20. ISBN: 978-3-88579-665-7 (cit. on p. 22).

[60] T. Benson, A. Akella, and D. A. Maltz. "Network Traffic Characteristics of Data Centers in the Wild." In: *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*. IMC '10. Melbourne, Australia: ACM, 2010, pp. 267–280. ISBN: 978-1-4503-0483-2. DOI: 10.1145/1879141.1879175. URL: http://doi.acm.org/10.1145/1879141.1879175 (cit. on pp. 35, 36).

[61] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita. "Towards Generating Real-life Datasets for Network Intrusion Detection." In: *International Journal of Network Security* 17 (2015), pp. 683–701 (cit. on p. 20).

[62] A. Biernacki. "Analysis of aggregated HTTP-based video traffic." In: *Journal of Communications and Networks* 18.5 (Oct. 2016), pp. 826–836. ISSN: 1229-2370. DOI: 10.1109/JCN.2016.000111 (cit. on pp. 30, 69, 70).

[63] E. Biglar Beigi, H. Hadian Jazi, N. Stakhanova, and A. A. Ghorbani. "Towards effective feature selection in machine learning-based botnet detection approaches." In: *2014 IEEE Conference on Communications and Network Security*. Oct. 2014, pp. 247–255. DOI: 10.1109/CNS.2014.6997492 (cit. on p. 21).

[64] J. Bispo, I. Sourdis, J. M. P. Cardoso, and S. Vassiliadis. "Regular expression matching for reconfigurable packet inspection." In: *2006 IEEE International Conference on Field Programmable Technology*. 2006. DOI: 10.1109/FPT.2006.270302 (cit. on p. 97).

[65] D. Borman, B. Braden, V. Jacobson, and R. Scheffenegger. *TCP Extensions for High Performance*. RFC 7323. http://www.rfc-editor.org/rfc/rfc7323.txt. 2014 (cit. on p. 49).

[66] A. Botta, A. Dainotti, and A. Pescapè. "A tool for the generation of realistic network workload for emerging networking scenarios." In: *Computer Networks* 56.15 (2012), pp. 3531–3547. DOI: 10.1016/j.comnet.2012.02.019 (cit. on p. 30).

[67] R. Braga, E. Mota, and A. Passito. "Lightweight DDoS flooding attack detection using NOX/OpenFlow." In: *IEEE 35th Local Computer Network Conference (LCN)*. Oct. 2010, pp. 408–415. DOI: 10.1109/LCN.2010.5735752 (cit. on pp. 142, 144, 152).

[68] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson. "TCP Vegas: New techniques for congestion detection and avoidance." In: *Proceedings of the Conference on Communications Architectures, Protocols and Applications (ACM SIGCOMM)*. 1994. DOI: 10.1145/190314.190317 (cit. on p. 44).

[69] D. Brauckhoff, B. Tellenbach, A. Wagner, M. May, and A. Lakhina. "Impact of Packet Sampling on Anomaly Detection Metrics." In: *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*. IMC '06. New York, NY, USA: ACM, 2006, pp. 159–164. ISBN: 1-59593-561-4. DOI: 10.1145/1177080.1177101. URL: http://doi.acm.org/10.1145/1177080.1177101 (cit. on p. 95).

[70] D. Brauckhoff, A. Wagner, and M. May. "FLAME: A Flow-level Anomaly Modeling Engine." In: *Proceedings of the Conference on Cyber Security Experimentation and Test*. CSET'08. San Jose, CA: USENIX Association, 2008, 1:1–1:6. URL: http://dl.acm.org/citation.cfm?id=1496662.1496663 (cit. on p. 31).

[71] L. Braun, C. Diekmann, N. Kammenhuber, and G. Carle. "Adaptive load-aware sampling for network monitoring on multicore commodity hardware." In: *IFIP Networking Conference, 2013*. 2013, pp. 1–9 (cit. on p. 95).

[72] L. Braun, A. Didebulidze, N. Kammenhuber, and G. Carle. "Comparing and Improving Current Packet Capturing Solutions Based on Commodity Hardware." In: *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*. IMC '10. Melbourne, Australia: ACM, 2010, pp. 206–217. ISBN: 978-1-4503-0483-2. DOI: 10.1145/1879141.1879168. URL: http://doi.acm.org/10.1145/1879141.1879168 (cit. on p. 116).

[73] S. M. Bridges and R. B. Vaughn. "Fuzzy Data Mining And Genetic Algorithms Applied To Intrusion Detection." In: *Proceedings of the National Information Systems Security Conference (NISSC)*. 2000, pp. 16–19 (cit. on p. 92).

[74] B. C. Brodie, D. E. Taylor, and R. K. Cytron. "A Scalable Architecture For High–Throughput Regular-Expression Pattern Matching." In: *33rd International Symposium on Computer Architecture (ISCA)*. 2006. DOI: 10.1109/ISCA.2006.7 (cit. on p. 97).

[75] C. Buragohain and N. Medhi. "FlowTrApp: An SDN based architecture for DDoS attack detection and mitigation in data centers." In: *2016 3rd International Conference on Signal Processing and Integrated Networks (SPIN)*. Feb. 2016, pp. 519–524. DOI: 10.1109/SPIN.2016.7566750 (cit. on pp. 142, 143, 152).

[76] H. Burch. "Tracing Anonymous Packets to Their Approximate Source." In: *Proceedings of the 14th USENIX Conference on System Administration*. LISA '00. New Orleans, Louisiana: USENIX Association, 2000, pp. 319–328. URL: http://dl.acm.org/citation.cfm?id=1045502.1045544 (cit. on p. 146).

[77] E. Bursztein, S. Bethard, C. Fabry, J. C. Mitchell, and D. Jurafsky. "How good are humans at solving CAPTCHAs? a large scale evaluation." In: *IEEE Symposium on Security and Privacy*. IEEE. 2010, pp. 399–413. DOI: 10.1109/SP.2010.31 (cit. on p. 175).

[78] M. Butkiewicz, H. V. Madhyastha, and V. Sekar. "Understanding Website Complexity: Measurements, Metrics, and Implications." In: *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*. IMC '11. Berlin, Germany: ACM, 2011, pp. 313–328. ISBN: 978-1-4503-1013-0. DOI: 10.1145/2068816.2068846. URL: http://doi.acm.org/10.1145/2068816.2068846 (cit. on pp. 30, 66, 67, 217).

[79] J. B. D. Cabrera, J. Gosar, W. Lee, and R. K. Mehra. "On the statistical distribution of processing times in network intrusion detection." In: *2004 43rd IEEE Conference on Decision and Control (CDC) (IEEE Cat. No.04CH37601)*. Vol. 1. Dec. 2004, 75–80 Vol.1. DOI: 10.1109/CDC.2004.1428609 (cit. on p. 99).

[80] J. B. D. Cabrera, L. Lewis, R. K. Prasanth, B. Ravichandran, and R. K. Mehra. "Proactive detection of distributed denial of service attacks using MIB traffic variables-a feasibility study." In: *2001 IEEE/IFIP International Symposium on Integrated Network Management Proceedings. Integrated Network Management VII. Integrated Management Strategies for the New Millennium (Cat. No.01EX470)*. May 2001, pp. 609–622. DOI: 10.1109/INM.2001.918069 (cit. on p. 146).

[81] R. Cáceres, P. B. Danzig, S. Jamin, and D. J. Mitzel. "Characteristics of Wide-area TCP/IP Conversations." In: SIGCOMM '91 (1991), pp. 101–112. DOI: 10.1145/115992.116003. URL: http://doi.acm.org/10.1145/115992.116003 (cit. on p. 30).

[82] A. O. Calchand, V. T. Dinh, P. Branch, and J. But. *BitTorrent Traffic Classification*. Tech. rep. 090227A. Melbourne, Australia: Centre for Advanced Internet Architectures, Swinburne University of Technology, 27 February 2009. URL: http://caia.swin.edu.au/reports/090227A/CAIA-TR-090227A.pdf (cit. on pp. 30, 68).

[83] C. Callegari, S. Giordano, and M. Pagano. "New statistical approaches for anomaly detection." In: *Security Comm. Networks* 2.6 (2009), pp. 611–634. ISSN: 1939-0122. DOI: 10.1002/sec.104. URL: http://onlinelibrary.wiley.com/doi/10.1002/sec.104/abstract (cit. on p. 90).

[84] S. Campbell and J. Lee. "Intrusion Detection at 100G." In: *State of the Practice Reports*. SC '11. New York, NY, USA: ACM, 2011. DOI: 10.1145/2063348.2063367. URL: http://doi.acm.org/10.1145/2063348.2063367 (cit. on p. 95).

[85] R. Casadesus-Masanell and A. Hervas-Drane. "Competing against online sharing." In: *Management Decision* 48.8 (2010), pp. 1247–1260. DOI: 10.1108/00251741011076771. eprint: https://doi.org/10.1108/00251741011076771. URL: https://doi.org/10.1108/00251741011076771 (cit. on p. 30).

[86] G. A. Cascallana and E. M. Lizarrondo. "Collecting packet traces at high speed." In: *In Proceedings of Workshop on Monitoring, Attack Detection and Mitigation*. 2006 (cit. on p. 116).

[87] Center for Applied Internet Data Analysis. *The CAIDA UCSD Anonymized Internet Traces 2015*. English. University of Californias San Diego Supercomputer Center. Dec. 2015. URL: http://www.caida.org/data/passive/passive_2015_dataset.xml (cit. on p. 24).

[88] D. Chaboya, R. Raines, R. Baldwin, and B. Mullins. "Network Intrusion Detection: Automated and Manual Methods Prone to Attack and Evasion." In: *IEEE Security & Privacy*. Vol. 4. 6. 2006, pp. 36–43. DOI: 10.1109/MSP.2006.159 (cit. on p. 94).

[89] B. Chandrasekaran. *Survey of Network Traffic Models*. http://www.cse.wustl.edu/~jain/cse567-06/traffic_models3.htm. Online; accessed 2018-01-15 (cit. on p. 29).

[90] A. Chandrasekhar and K. Raghuveer. "Confederation of FCM clustering, ANN and SVM techniques to implement hybrid NIDS using corrected KDD cup 99 dataset." In: *Communications and Signal Processing (ICCSP), 2014 International Conference on*. Apr. 2014, pp. 672–676. DOI: 10.1109/ICCSP.2014.6949927 (cit. on p. 22).

[91] C. Chen, Y. Chen, W. Lu, S. Tsai, and M. Yang. "Detecting amplification attacks with Software Defined Networking." In: *2017 IEEE Conference on Dependable and Secure Computing*. Aug. 2017, pp. 195–201. DOI: 10.1109/DESEC.2017.8073807 (cit. on pp. 142, 145, 152).

[92] M.-H. Chen, J.-Y. Ciou, I.-H. Chung, and C.-F. Chou. "FlexProtect: A SDN-based DDoS Attack Protection Architecture for Multi-tenant Data Centers." In: *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region*. HPC Asia 2018. Chiyoda, Tokyo, Japan: ACM, 2018, pp. 202–209. ISBN: 978-1-4503-5372-4. DOI: 10.1145/3149457.3149476. URL: http://doi.acm.org/10.1145/3149457.3149476 (cit. on pp. 142, 143, 152).

[93] P. J. Chen and Y. W. Chen. "Implementation of SDN based network intrusion detection and prevention system." In: *International Carnahan Conference on Security Technology (ICCST)*. Sept. 2015, pp. 141–146. DOI: 10.1109/CCST.2015.7389672 (cit. on p. 147).

[94] R. Chen, J. Park, and R. Marchany. "RIM: Router Interface Marking for IP Traceback." In: *IEEE Globecom 2006*. Nov. 2006, pp. 1–5. DOI: 10.1109/GLOCOM.2006.312 (cit. on p. 146).

[95] R. Chen and J.-M. Park. "Attack diagnosis: throttling distributed denial-of-service attacks close to the attack sources." In: *Proceedings. 14th International Conference on Computer Communications and Networks, 2005. ICCCN 2005.* Oct. 2005, pp. 275–280. DOI: 10.1109/ICCCN.2005.1523866 (cit. on p. 147).

[96] T. M. Chen. "Network Traffic Modeling." In: *Handbook of Computer Networks*. Wiley-Blackwell, 2012, pp. 326–339. ISBN: 9781118256107. DOI: 10.1002/9781118256107.ch21. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781118256107.ch21. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118256107.ch21 (cit. on p. 29).

[97] T.-H. Cheng, Y.-D. Lin, Y.-C. Lai, and P.-C. Lin. "Evasion Techniques: Sneaking through Your Intrusion Detection/Prevention Systems." In: *Communications Surveys Tutorials, IEEE* 14.4

(2012), pp. 1011–1020. ISSN: 1553-877X. DOI: 10.1109/SURV.2011.092311.00082 (cit. on p. 94).

[98] H.-K. Choi and J. O. Limb. "A behavioral model of Web traffic." In: *Proceedings. Seventh International Conference on Network Protocols*. Oct. 1999, pp. 327–334. DOI: 10.1109/ICNP.1999.801961 (cit. on pp. 30, 66, 67, 217).

[99] Cisco. *The Zettabyte Era: Trends and Analysis*. https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.html. Online; accessed 2018-02-07 (cit. on p. 219).

[100] C. R. Clark and D. E. Schimmel. "Scalable pattern matching for high speed networks." In: *12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 2004. DOI: 10.1109/FCCM.2004.50 (cit. on p. 97).

[101] Cloudflare. *GoFlow Documentation*. https://github.com/cloudflare/goflow/tree/version3. Online; accessed: 2019-11-18 (cit. on pp. 223, 224).

[102] C. G. Cordero, E. Vasilomanolakis, N. Milanov, C. Koch, D. Hausheer, and M. Mühlhäuser. "ID2T: A DIY dataset creation toolkit for Intrusion Detection Systems." In: *2015 IEEE Conference on Communications and Network Security (CNS)*. Sept. 2015, pp. 739–740. DOI: 10.1109/CNS.2015.7346912 (cit. on p. 31).

[103] G. Creech. "Developing a high-accuracy cross platform Host-Based Intrusion Detection System capable of reliably detecting zero-day attacks." PhD thesis. UNSW Canberra, 2014 (cit. on pp. 21, 22).

[104] G. Creech and J. Hu. "Generation of a new IDS test dataset: Time to retire the KDD collection." In: *Wireless Communications and Networking Conference (WCNC), 2013 IEEE*. Apr. 2013, pp. 4487–4492. DOI: 10.1109/WCNC.2013.6555301 (cit. on p. 21).

[105] G. Creech and J. Hu. "A Semantic Approach to Host-Based Intrusion Detection Systems Using Contiguousand Discontiguous System Call Patterns." In: *Computers, IEEE Transactions on* 63.4 (Apr. 2014), pp. 807–819. ISSN: 0018-9340. DOI: 10.1109/TC.2013.13 (cit. on p. 21).

[106] Y. Cui, L. Yan, S. Li, H. Xing, W. Pan, J. Zhu, and X. Zheng. "SD-Anti-DDoS: Fast and efficient DDoS defense in software-defined networks." In: *Journal of Network and Computer Applications* 68 (2016), pp. 65–79. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2016.04.005 (cit. on pp. 142, 145, 152).

[107] M. V. O. De Assis, A. H. Hamamoto, T. Abrão, and M. L. Proença. "A Game Theoretical Based System Using Holt-Winters and Genetic Algorithm With Fuzzy Logic for DoS/DDoS Mitigation on SDN Networks." In: *IEEE Access* 5 (2017), pp. 9485–9496. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2017.2702341 (cit. on pp. 142, 145, 152).

[108] D. Denning and P. Neumann. *Requirements and model for IDES – a real-time intrusion detection system*. Tech. rep. Technical Report. Computer Science Laboratory, SRI International, 1985 (cit. on p. 90).

[109] L. Deri, N. S. P. A, V. D. B. Km, and L. L. Figuretta. "Improving Passive Packet Capture: Beyond Device Polling." In: *In Proceedings of SANE*. 2004 (cit. on p. 116).

[110] T. Detristan, T. Ulenspiegel, Y. Malcom, and M. Superbus von Underduk. *Polymorphic Shellcode Engine Using Spectrum Analysis*. URL: http://phrack.org/issues/61/9.html (visited on 01/08/2020) (cit. on p. 94).

[111] Deutsches Forschungsnetz. "DDoS-Schutz 2.0 – Der Regelbetrieb beginnt." In: *DFN Mitteilungen*. Nov. 2016, pp. 40–45 (cit. on p. 149).

[112] Deutsches Forschungsnetz. "NeMo: Die Technik hinter der DoS-Abwehrplattform im X-WiN." In: *DFN Mitteilungen*. Nov. 2016, pp. 46–49 (cit. on pp. 149, 152).

[113] J. E. Dickerson and J. A. Dickerson. "Fuzzy network profiling for intrusion detection." In: *19th International Conference of the North American Fuzzy Information Processing Society NAFIPS*. 2000, pp. 301–306. DOI: 10.1109/NAFIPS.2000.877441 (cit. on p. 92).

[114] C. Dumitrescu, I. Raicu, M. Ripeanu, and I. Foster. "DiPerF: an automated distributed performance testing framework." In: *Fifth IEEE/ACM International Workshop on Grid Computing*. Nov. 2004, pp. 289–296. DOI: 10.1109/GRID.2004.21 (cit. on p. 32).

[115] B. Al-Duwairi and M. Govindarasu. "Novel hybrid schemes employing packet marking and logging for IP traceback." In: *IEEE Transactions on Parallel and Distributed Systems* 17.5 (May 2006), pp. 403–418. ISSN: 1045-9219. DOI: 10.1109/TPDS.2006.63 (cit. on p. 146).

[116] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, and G. Carle. "MoonGen: A Scriptable High-Speed Packet Generator." In: *Proceedings of the 2015 Internet Measurement Conference*. IMC '15. Tokyo, Japan: ACM, 2015, pp. 275–287. ISBN: 978-1-4503-3848-6. DOI: 10.1145/2815675.2815692. URL: http://doi.acm.org/10.1145/2815675.2815692 (cit. on p. 31).

[117] F. Erlacher and F. Dressler. "How to Test an IDS?: GENESIDS: An Automated System for Generating Attack Traffic." In: *Proceedings of the 2018 Workshop on Traffic Measurements for Cybersecurity*. WTMC '18. Budapest, Hungary: ACM, 2018, pp. 46–51. ISBN: 978-1-4503-5910-8. DOI: 10.1145/3229598.3229601. URL: http://doi.acm.org/10.1145/3229598.3229601 (cit. on pp. 31, 80).

[118] R. Ferguson and D. Senie. *Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing*. RFC 2827 (Best Current Practice). http://www.rfc-editor.org/rfc/rfc2827.txt Updated by RFC 3704. Internet Engineering Task Force, May 2000 (cit. on pp. 141, 146).

[119] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1*. RFC 2616 (Draft Standard). http://www.rfc-editor.org/rfc/rfc2616.txt. Internet Engineering Task Force, June 1999 (cit. on pp. 137, 158).

[120] A. Fießler, S. Hager, B. Scheuermann, and A. von Gernler. "HardFIRE - ein Firewall-Konzept auf FPGA-Basis." In: *14. Deutscher IT-Sicherheitskongress*. 2015 (cit. on p. 98).

[121] R. W. Floyd and J. D. Ullman. "The compilation of regular expressions into integrated circuits." In: *IEEE FOCS*. 1980. DOI: 10.1109/SFCS.1980.44 (cit. on p. 97).

[122] S. Floyd. *HighSpeed TCP for Large Congestion Windows*. RFC 3649. http://www.rfc-editor.org/rfc/rfc3649.txt. 2003 (cit. on p. 45).

[123] S. Floyd, T. Henderson, and A. Gurtov. *The NewReno Modification to TCP's Fast Recovery Algorithm*. RFC 3782. http://www.rfc-editor.org/rfc/rfc3782.txt. 2004 (cit. on p. 45).

[124] A. Gaikwad and R. Jaiswal. "Experimental Analysis of Bittorrent Traffic based on Heavy-Tailed Probability Distributions." In: *International Journal of Computer Applications* 155.2 (2016). DOI: 10.5120/ijca2016912268 (cit. on p. 30).

[125] B. Galloway and G. Hancke. "Introduction to Industrial Control Networks." In: *Communications Surveys Tutorials, IEEE* 15.2 (Feb. 2013), pp. 860–880. ISSN: 1553-877X. DOI: 10.1109/SURV.2012.071812.00124 (cit. on p. 36).

[126] Y. Gao, Y. Feng, J. Kawamoto, and K. Sakurai. "A Machine Learning Based Approach for Detecting DRDoS Attacks and Its Performance Evaluation." In: *AsiaJCIS*. IEEE. 2016, pp. 80–86. DOI: 10.1109/AsiaJCIS.2016.24 (cit. on pp. 142–144, 152).

[127] S. Garcia, M. Grill, J. Stiborek, and A. Zunino. "An Empirical Comparison of Botnet Detection Methods." In: *Comput. Secur.* 45 (Sept. 2014), pp. 100–123. ISSN: 0167-4048. DOI: 10.1016/j.cose.2014.05.011. URL: http://dx.doi.org/10.1016/j.cose.2014.05.011 (cit. on p. 26).

[128] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, and E. Vázquez. "Anomaly-based network intrusion detection: Techniques, systems and challenges." In: *Computers & Security* 28.1 (2009), pp. 18–28. ISSN: 0167-4048. DOI: 10.1016/j.cose.2008.08.003. URL: http://www.sciencedirect.com/science/article/pii/S0167404808000692 (cit. on pp. 90, 92–94).

[129] C. Gero. *Moving Beyond Perimeter Security—A Comprehensive and Achievable Guide to Less Risk*. https://content.akamai.com/us-en-PG10736-zero-trust-moving-beyond-perimeter-security.html. Online; accessed 2019-06-06 (cit. on p. 3).

[130] T. M. Gil and M. Poletto. "MULTOPS: A Data-structure for Bandwidth Attack Detection." In: *Proceedings of the 10th Conference on USENIX Security Symposium - Volume 10*. SSYM'01. Washington, D.C.: USENIX Association, 2001, pp. 3–3. URL: http://dl.acm.org/citation.cfm?id=1267612.1267615 (cit. on p. 141).

[131] G. Gilder. "TELECOSM: How Infinite Bandwidth Will Revolutionize Our World." In: *The Free Press* (2000) (cit. on pp. 4, 99).

[132] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras, and V. Maglaris. "Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments." In: *Computer Networks: The International Journal of Computer and Telecommunications Networking* 62 (2014), pp. 122–136. DOI: 10.1016/j.bjp.2013.10.014 (cit. on p. 143).

[133] P. Gogoi, M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita. "Packet and Flow Based Network Intrusion Dataset." In: *Contemporary Computing*. Ed. by M. Parashar, D. Kaushik, O. F. Rana, R. Samtaney, Y. Yang, and A. Zomaya. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 322–334. ISBN: 978-3-642-32129-0 (cit. on p. 20).

[134] J. M. Gonzalez, V. Paxson, and N. Weaver. "Shunting: A Hardware/Software Architecture for Flexible, High-performance Network Intrusion Prevention." In: *Proceedings of the 14th ACM Conference on Computer and Communications Security*. CCS '07. New York, NY, USA: ACM, 2007, pp. 139–149. ISBN: 978-1-59593-703-2. DOI: 10.1145/1315245.1315264. URL: http://doi.acm.org/10.1145/1315245.1315264 (cit. on pp. 95, 97).

[135] F. Gringoli, L. Salgarelli, M. Dusi, N. Cascarano, F. Risso, and K. Claffy. "GT: picking up the truth from the ground for Internet traffic." In: *ACM SIGCOMM Computer Communication Review (CCR)* (Oct. 2009). DOI: 10.1145/1629607.1629610 (cit. on p. 25).

[136] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan. "Measurement, Modeling, and Analysis of a Peer-to-peer File-sharing Workload." In: *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*. SOSP '03. Bolton Landing, NY, USA: ACM, 2003, pp. 314–329. ISBN: 1-58113-757-5. DOI: 10.1145/945445.945475. URL: http://doi.acm.org/10.1145/945445.945475 (cit. on p. 30).

[137] S. Ha, I. Rhee, and L. Xu. "CUBIC: a new TCP-friendly high-speed TCP variant." In: *ACM SIGOPS Operating Systems Review - Research and developments in the Linux kernel* Volume 42 Issue 5 (2008), 64–74. DOI: 10.1145/1400097.1400105 (cit. on p. 46).

[138] S. Ha, I. Rhee, and L. Xu. "Comparison of High Speed Congestion Control Protocols." In: *International Journal of Network Security & Its Applications (IJNSA)* Volume 4 Issue 5 (2012) (cit. on pp. 37, 38, 44).

[139] S. Hager, F. Winkler, B. Scheuermann, and K. Reinhardt. "Building Optimized Packet Filters with COFFi." In: *IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. May 2014. DOI: 10.1109/FCCM.2014.38 (cit. on p. 98).

[140] S. Hager, F. Winkler, B. Scheuermann, and K. Reinhardt. "MPFC: Massively Parallel Firewall Circuits." In: *39th IEEE Conference on Local Computer Networks (LCN)*. Sept. 2014. DOI: 10.1109/LCN.2014.6925785 (cit. on p. 98).

[141] J. Haines, L. Rossey, R. Lippmann, and R. Cunningham. "Extending the DARPA off-line intrusion detection evaluations." In: *DARPA Information Survivability Conference amp; Exposition II, 2001. DISCEX '01. Proceedings*. Vol. 1. 2001, 35–45 vol.1. DOI: 10.1109/DISCEX.2001.932190 (cit. on p. 19).

[142] A. Harbola, J. Harbola, and K. Vaisla. "Improved Intrusion Detection in DDoS Applying Feature Selection Using Rank amp;amp; Score of Attributes in KDD-99 Data Set." In: *Computational Intelligence and Communication Networks (CICN), 2014 International Conference on*. Nov. 2014, pp. 840–845. DOI: 10.1109/CICN.2014.179 (cit. on p. 22).

[143] D. He, S. Chan, X. Ni, and M. Guizani. "Software-Defined-Networking-Enabled Traffic Anomaly Detection and Mitigation." In: *IEEE Internet of Things Journal* 4.6 (Dec. 2017),

pp. 1890–1898. ISSN: 2327-4662. DOI: 10.1109/JIOT.2017.2694702 (cit. on pp. 142, 145, 152).

[144] T. Hirakawa, K. Ogura, B. B. Bista, and T. Takata. "A Defense Method against Distributed Slow HTTP DoS Attack." In: *19th International Conference on Network-Based Information Systems (NBiS)*. IEEE. 2016, pp. 152–158. DOI: 10.1109/NBiS.2016.58 (cit. on pp. 145, 152).

[145] M. Hock, R. Bless, and M. Zitterbart. "Experimental evaluation of BBR congestion control." In: *2017 IEEE 25th International Conference on Network Protocols (ICNP)*. Oct. 2017, pp. 1–10. DOI: 10.1109/ICNP.2017.8117540 (cit. on p. 39).

[146] M. Hock, M. Veit, F. Neumeister, R. Bless, and M. Zitterbart. "TCP at 100Gbit/s - Tuning, Limitations, Congestion Control." In: 44th IEEE Conference on Local Computer Networks (LCN). 2019. ISBN: 978-1-7281-1028-8 (cit. on pp. 7, 44).

[147] R. Hofstede, L. Hendriks, A. Sperotto, and A. Pras. "SSH Compromise Detection Using NetFlow/IPFIX." In: *SIGCOMM Comput. Commun. Rev.* 44.5 (Oct. 2014), pp. 20–26. ISSN: 0146-4833. DOI: 10.1145/2677046.2677050. URL: http://doi.acm.org/10.1145/2677046.2677050 (cit. on p. 27).

[148] K. Hong, Y. Kim, H. Choi, and J. Park. "SDN-Assisted Slow HTTP DDoS Attack Defense Method." In: *IEEE Communications Letters* PP.99 (2017), pp. 1–1. ISSN: 1089-7798. DOI: 10.1109/LCOMM.2017.2766636 (cit. on pp. 148, 152, 168).

[149] M. S. Hoque, M. A. Mukit, and M. A. N. Bikas. "An Implementation of Intrusion Detection System Using Genetic Algorithm." In: *International Journal of Network Security & Its Applications* 4.2 (2012), pp. 109–120. ISSN: 09752307. DOI: 10.5121/ijnsa.2012.4208. arXiv: 1204.1336. URL: http://arxiv.org/abs/1204.1336 (cit. on p. 92).

[150] D. Hu, P. Hong, and Y. Chen. "FADM: DDoS Flooding Attack Detection and Mitigation System in Software-Defined Networking." In: *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*. Dec. 2017, pp. 1–7. DOI: 10.1109/GLOCOM.2017.8254023 (cit. on pp. 142, 145, 152).

[151] B. L. Hutchings, R. Franklin, and D. Carver. "Assisting network intrusion detection with reconfigurable hardware." In: *10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 2002. DOI: 10.1109/FPGA.2002.1106666 (cit. on p. 97).

[152] IEEE. "Standard for Information technology - Local and metropolitan area networks - Part 3: CSMA/CD Access Method and Physical Layer Specifications - Media Access Control (MAC) Parameters, Physical Layer, and Management Parameters for

10 Gb/s Operation." In: *IEEE Std. 802.3ae-2002* (2002) (cit. on p. 55).

[153] Imperva. *Global DDoS Threat Landscape Q4 2017*. https://www.imperva.com/resources/resource-library/reports/q4-2017-global-ddos-threat-landscape/. Online; accessed 2019-05-02 (cit. on pp. 138, 162).

[154] V. Jacobson and R. Braden. *TCP Extensions for Long-Delay Paths*. RFC 1072. http://www.rfc-editor.org/rfc/rfc1072.txt. Mar. 2013. DOI: 10.17487/rfc1072 (cit. on p. 43).

[155] K. Jaic, M. C. Smith, and N. Sarma. "A practical network intrusion detection system for inline FPGAs on 10GbE network adapters." In: *2014 IEEE 25th International Conference on Application-Specific Systems, Architectures and Processors*. June 2014, pp. 180–181. DOI: 10.1109/ASAP.2014.6868655 (cit. on p. 98).

[156] R. Jain, D. Chiu, and W. Haw. *A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer System*. Tech. rep. Hudson, USA: Digital Equipment Corporation, 1984 (cit. on p. 38).

[157] R. Jalili, F. Imani-Mehr, M. Amini, and H. R. Shahriari. "Detection of Distributed Denial of Service Attacks Using Statistical Pre-processor and Unsupervised Neural Networks." In: *Information Security Practice and Experience*. Ed. by R. H. Deng, F. Bao, H. Pang, and J. Zhou. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 192–203. ISBN: 978-3-540-31979-5 (cit. on p. 146).

[158] W. Jiang and V. K. Prasanna. "Field-split Parallel Architecture for High Performance Multi-match Packet Classification Using FPGAs." In: *Proceedings of the Twenty-first Annual Symposium on Parallelism in Algorithms and Architectures*. SPAA '09. New York, NY, USA: ACM, 2009, pp. 188–196. ISBN: 978-1-60558-606-9. DOI: 10.1145/1583991.1584044. URL: http://doi.acm.org/10.1145/1583991.1584044 (cit. on p. 96).

[159] A. John and T. Sivakumar. "DDoS: Survey of Traceback Methods." In: *International Journal of Recent Trends in Engineering* (May 2009) (cit. on p. 146).

[160] W. John. "Characterization and Classification of Internet Backbone Traffic." PhD thesis. 2010. ISBN: 978-91-7385-363-7 (cit. on p. 36).

[161] M. Jonker. "DDoS Mitigation: A Measurement-Based Approach." PhD thesis. Netherlands: University of Twente, Oct. 2019. ISBN: 978-90-365-4868-7. DOI: 10.3990/1.9789036548687 (cit. on pp. 140, 152).

[162] M. Jonker, A. King, J. Krupp, C. Rossow, A. Sperotto, and A. Dainotti. "Millions of Targets Under Attack: a Macroscopic Characterization of the DoS Ecosystem." In: *ACM Internet Measurement Conference (IMC)*. Nov. 2017. DOI: 10.1145/3131365.3131383 (cit. on pp. 136, 138, 171).

[163] M. Jonker, A. Sperotto, R. van Rijswijk-Deij, R. Sadre, and A. Pras. "Measuring the Adoption of DDoS Protection Services." In: *ACM Internet Measurement Conference (IMC)*. Santa Monica, California, USA: ACM, 2016, pp. 279–285. DOI: 10.1145/2987443.2987487 (cit. on pp. 140, 161).

[164] G. Kambourakis, T. Moschos, D. Geneiatakis, and S. Gritzalis. "Detecting DNS Amplification Attacks." In: *Critical Information Infrastructures Security*. Ed. by J. Lopez and B. M. Hämmerli. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 185–196. ISBN: 978-3-540-89173-4 (cit. on p. 146).

[165] P. Kampanakis, H. Perros, and T. Beyene. "SDN-based solutions for Moving Target Defense network protection." In: *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*. June 2014, pp. 1–6. DOI: 10.1109/WoWMoM.2014.6918979 (cit. on p. 147).

[166] S. Kandula, D. Katabi, M. Jacob, and A. Berger. "Botz-4-sale: Surviving Organized DDoS Attacks That Mimic Flash Crowds." In: *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*. NSDI'05. Berkeley, CA, USA: USENIX Association, 2005, pp. 287–300. URL: http://dl.acm.org/citation.cfm?id=1251203.1251224 (cit. on p. 148).

[167] M. Al-kasassbeh, G. Al-Naymat, A. Hassanat, and M. Almseidin. "Detecting Distributed Denial of Service Attacks Using Data Mining Techniques." In: *International Journal of Advanced Computer Science and Applications (IJACSA)* (2016) (cit. on p. 21).

[168] Kaspersky Lab. *DDoS attacks in Q1 2019*. https://securelist.com/ddos-report-q1-2019/90792/. Online; accessed 2019-06-05 (cit. on p. 135).

[169] T. Kelly. "Scalable TCP: Improving Performance in Highspeed Wide Area Networks." In: *ACM SIGOPS Operating Systems Review* Volume 33 Issue 2 (2003), 83–91. DOI: 10.1145/956981.956989 (cit. on p. 45).

[170] A. D. Kent. *Comprehensive, Multi-Source Cyber-Security Events*. Los Alamos National Laboratory. 2015. DOI: 10.17021/1179829 (cit. on p. 25).

[171] F. Khan, M. Gokhale, and C.-N. Chuah. "FPGA Based Network Traffic Analysis Using Traffic Dispersion Patterns." In: *2010 International Conference on Field Programmable Logic and Applications (FPL)*. 2010 International Conference on Field Programmable Logic and Applications (FPL). 2010, pp. 519–524. DOI: 10.1109/FPL.2010.103 (cit. on p. 96).

[172] H. Kholidy and F. Baiardi. "CIDD: A Cloud Intrusion Detection Dataset for Cloud Computing and Masquerade Attacks." In: *Information Technology: New Generations (ITNG), 2012 Ninth International Conference on*. Apr. 2012, pp. 397–402. DOI: 10.1109/ITNG.2012.97 (cit. on p. 28).

[173] M. Kim, Y. Won, H. Lee, J. Hong, and R. Boutaba. *Flow-based Characteristic Analysis of Internet Application Traffic*. Tech. rep. 2004 (cit. on pp. 30, 71, 220).

[174] Y. Kim, W. C. Lau, M. C. Chuah, and H. J. Chao. "PacketScore: a statistics-based packet filtering scheme against distributed denial-of-service attacks." In: *IEEE Transactions on Dependable and Secure Computing* 3.2 (Apr. 2006), pp. 141–155. ISSN: 2160-9209. DOI: 10.1109/TDSC.2006.25 (cit. on pp. 142, 146).

[175] J. Kindervag. "No more chewy centers: Introducing the zero trust model of information security." In: *Forrester Research* (2010) (cit. on p. 3).

[176] V. V. Kindratenko, J. J. Enos, G. Shi, M. T. Showerman, G. W. Arnold, J. E. Stone, J. C. Phillips, and W. Hwu. "GPU clusters for high-performance computing." In: *2009 IEEE International Conference on Cluster Computing and Workshops*. Aug. 2009, pp. 1–8. DOI: 10.1109/CLUSTR.2009.5289128 (cit. on p. 120).

[177] E. Kline, M. Beaumont-Gay, J. Mirkovic, and P. Reiher. "RAD: Reflector Attack Defense Using Message Authentication Codes." In: *2009 Annual Computer Security Applications Conference*. Dec. 2009, pp. 269–278 (cit. on p. 147).

[178] C. Kolias, G. Kambourakis, A. Stavrou, and S. Gritzalis. "Intrusion Detection in 802.11 Networks: Empirical Evaluation of Threats and a Public Dataset." In: *Communications Surveys Tutorials, IEEE* PP.99 (2015), pp. 1–1. ISSN: 1553-877X. DOI: 10.1109/COMST.2015.2402161 (cit. on p. 28).

[179] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas. "DDoS in the IoT: Mirai and Other Botnets." In: *Computer* 50.7 (2017), pp. 80–84. ISSN: 0018-9162. DOI: 10.1109/MC.2017.201 (cit. on p. 133).

[180] B. Krebs. *New Mirai Worm Knocks 900K Germans Offline*. https://krebsonsecurity.com/2016/11/new-mirai-worm-knocks-900k-germans-offline/. Online; accessed 2019-05-02 (cit. on pp. 99, 133).

[181] M. Kumar. "1.7 Tbps DDoS Attack – Memcached UDP Reflections Set New Record." In: (Online; accessed on 2018-04-02). URL: https://thehackernews.com/2018/03/ddos-attack-memcached.html (cit. on p. 138).

[182] A. Lakhina, M. Crovella, and C. Diot. "Mining Anomalies Using Traffic Feature Distributions." In: *Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. SIGCOMM '05. New York, NY, USA: ACM, 2005, pp. 217–228. ISBN: 1-59593-009-4. DOI: 10.1145/1080091.1080118. URL: http://doi.acm.org/10.1145/1080091.1080118 (cit. on p. 90).

[183] M. Laterman, M. Arlitt, and C. Williamson. "A campus-level view of Netflix and Twitch: Characterization and performance implications." In: *2017 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*. July 2017, pp. 1–8. DOI: 10.23919/SPECTS.2017.8046774 (cit. on pp. 30, 70, 219).

[184] C. Lavole. *Application-Layer DDoS Attack Protection with HAProxy*. https://www.haproxy.com/blog/application-layer-ddos-attack-protection-with-haproxy/. Online; accessed 2019-05-13 (cit. on p. 149).

[185] T. M. Le and J. But. *Bittorrent traffic classification*. Tech. rep. 091022A. Melbourne, Australia: Centre for Advanced Internet Architectures, Swinburne University of Technology, 22 October 2009. URL: http://caia.swin.edu.au/reports/091022A/CAIA-TR-091022A.pdf (cit. on pp. 30, 68, 218).

[186] C. Leckie and K. Ramamohanarao. "Protection from distributed denial of service attacks using history-based IP filtering." In: *IEEE International Conference on Communications, 2003. ICC '03.* Vol. 1. May 2003, 482–486 vol.1. DOI: 10.1109/ICC.2003.1204223 (cit. on p. 143).

[187] C. B. Lee, C. Roedel, and E. Silenok. *Detection and Characterization of Port Scan Attacks*. Tech. rep. 2003 (cit. on p. 71).

[188] J.-H. Lee, J.-H. Lee, S.-G. Sohn, J.-H. Ryu, and T.-M. Chung. "Effective Value of Decision Tree with KDD 99 Intrusion Detection Datasets for Intrusion Detection System." In: *Advanced Communication Technology, 2008. ICACT 2008. 10th International Conference on*. Vol. 2. Feb. 2008, pp. 1170–1175. DOI: 10.1109/ICACT.2008.4493974 (cit. on p. 22).

[189] S. Lee, J. Kim, S. Shin, P. Porras, and V. Yegneswaran. "Athena: A Framework for Scalable Anomaly Detection in Software-Defined Networks." In: *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. June 2017, pp. 249–260. DOI: 10.1109/DSN.2017.42 (cit. on pp. 142, 145, 152).

[190] B. H. Leitao. "Tuning 10Gb network cards on Linux." In: *Proceedings of the Ottawa Linux Symposium (OLS)*. 2009 (cit. on p. 44).

[191] D. Leith and R. Shorten. "H-TCP: TCP for high-speed and long-distance networks." In: *Proceedings of the PFLDnet Argonne*. 2004 (cit. on p. 45).

[192] C. Li, Y. Wu, X. Yuan, Z. Sun, W. Wang, X. Li, and L. Gong. "Detection and defense of DDoS attack–based on deep learning in OpenFlow-based SDN." In: *International Journal of Communication Systems* 31.5 (2018). e3497 IJCS-17-0848.R1, e3497. DOI: 10.1002/dac.3497. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/dac.3497. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/dac.3497 (cit. on pp. 142, 144, 152).

[193] M. Li, J. Liu, and D. Long. "Probability Principle of a Reliable Approach to Detect Signs of DDOS Flood Attacks." In: *Parallel and Distributed Computing: Applications and Technologies*. Ed. by K.-M. Liew, H. Shen, S. See, W. Cai, P. Fan, and S. Horiguchi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 596–599. ISBN: 978-3-540-30501-9 (cit. on p. 146).

[194] W. Li. "Using Genetic Algorithm for network intrusion detection." In: *In Proceedings of the United States Department of Energy Cyber Security Group Training Conference*. 2004, pp. 24–27 (cit. on p. 92).

[195] Y. T. Li, D. Leith, and R. N. Shorten. "Experimental Evaluation of TCP Protocols for High-Speed Networks." In: *IEEE/ACM Transactions on Networking* 15.5 (Oct. 2007), pp. 1109–1122. ISSN: 1063-6692. DOI: 10.1109/TNET.2007.896240 (cit. on pp. 37, 38, 47, 61).

[196] S. Lim, J. Ha, H. Kim, Y. Kim, and S. Yang. "A SDN-oriented DDoS blocking scheme for botnet-based attacks." In: *Sixth International Conference on Ubiquitous and Future Networks*. July 2014, pp. 63–68. DOI: 10.1109/ICUFN.2014.6876752 (cit. on pp. 147, 169).

[197] C.-H. Lin, C.-T. Huang, C.-P. Jiang, and S.-C. Chang. "Optimization of Regular Expression Pattern Matching Circuits on FPGA." In: *Proceedings of the Design Automation & Test in Europe Conference*. 2006 (cit. on p. 97).

[198] R. Lippmann, D. Fried, I. Graf, J. Haines, K. Kendall, D. McClung, D. Weber, S. Webster, D. Wyschogrod, R. Cunningham, and M. Zissman. "Evaluating intrusion detection systems: the 1998 DARPA off-line intrusion detection evaluation." In: *DARPA Information Survivability Conference and Exposition, 2000. DISCEX '00. Proceedings*. Vol. 2. 2000, 12–26 vol.2. DOI: 10.1109/DISCEX.2000.821506 (cit. on p. 19).

[199] J. Liu, Y. Lai, and S. Zhang. "FL-GUARD: A Detection and Defense System for DDoS Attack in SDN." In: *Proceedings of the 2017 International Conference on Cryptography, Security and Privacy*. ICCSP '17. Wuhan, China: ACM, 2017, pp. 107–111. ISBN: 978-1-4503-4867-6. DOI: 10.1145/3058060.3058074. URL: http://doi.acm.org/10.1145/3058060.3058074 (cit. on pp. 142, 145).

[200] D. Luchaup, R. Smith, C. Estan, and S. Jha. "Multi-byte Regular Expression Matching with Speculation." In: *Recent Advances in Intrusion Detection*. Ed. by E. Kirda, S. Jha, and D. Balzarotti. Lecture Notes in Computer Science 5758. Springer Berlin Heidelberg, 2009, pp. 284–303. ISBN: 978-3-642-04341-3, 978-3-642-04342-0. URL: http://link.springer.com/chapter/10.1007/978-3-642-04342-0%5C_15 (cit. on p. 97).

[201] G. Maciá-Fernández, J. Camacho, R. Magán-Carrión, P. García-Teodoro, and R. Therón. "UGR'16: A new dataset for the evaluation of cyclostationarity-based network IDSs." In: *Computers & Security* 73 (2018), pp. 411–424. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2017.11.004. URL: http://www.sciencedirect.com/science/article/pii/S0167404817302353 (cit. on p. 27).

[202] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. "Controlling High Bandwidth Aggregates in the Network." In: *SIGCOMM Comput. Commun. Rev.* 32.3 (July 2002), pp. 62–73. ISSN: 0146-4833. DOI: 10.1145/571697.571724. URL: http://doi.acm.org/10.1145/571697.571724 (cit. on p. 147).

[203] A. Mahimkar, J. Dange, V. Shmatikov, H. M. Vin, and Y. Zhang. "dFence: Transparent Network-based Denial of Service Mitigation." In: *4th USENIX Symposium on Networked Systems Design & Implementation (NSDI)*. Vol. 7. 2007 (cit. on p. 148).

[204] M. V. Mahoney and P. K. Chan. "An Analysis of the 1999 DARPA/Lincoln Laboratory Evaluation Data for Network Anomaly Detection." In: *Recent Advances in Intrusion Detection*. Ed. by G. Vigna, C. Kruegel, and E. Jonsson. Lecture Notes in Computer Science 2820. Springer Berlin Heidelberg, 2003, pp. 220–237. URL: http://link.springer.com/chapter/10.1007/978-3-540-45248-5_13 (cit. on pp. 22, 91).

[205] J. Mai, C.-N. Chuah, A. Sridharan, T. Ye, and H. Zang. "Is Sampled Data Sufficient for Anomaly Detection?" In: *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*. IMC '06. New York, NY, USA: ACM, 2006, pp. 165–176. ISBN: 1-59593-561-4. DOI: 10.1145/1177080.1177102. URL: http://doi.acm.org/10.1145/1177080.1177102 (cit. on p. 95).

[206] J. Mai, A. Sridharan, C.-N. Chuah, H. Zang, and T. Ye. "Impact of Packet Sampling on Portscan Detection." In: *IEEE Journal on Selected Areas in Communications* 24.12 (Dec. 2006), pp. 2285–2298. ISSN: 0733-8716. DOI: 10.1109/JSAC.2006.884027 (cit. on p. 95).

[207] P. Manandhar and Z. Aung. "Towards Practical Anomaly-Based Intrusion Detection by Outlier Mining on TCP Packets." In: *Database and Expert Systems Applications*. Ed. by H. Decker, L. Lhotská, S. Link, M. Spies, and R. R. Wagner. Lecture Notes in Computer Science 8645. Springer International Publishing, 2014, pp. 164–173. ISBN: 978-3-319-10084-5, 978-3-319-10085-2. URL: http://link.springer.com/chapter/10.1007/978-3-319-10085-2%5C_14 (cit. on p. 92).

[208] M. Mantere, M. Sailio, and S. Noponen. "Network Traffic Features for Anomaly Detection in Specific Industrial Control System Network." In: *Future Internet* 5.4 (2013), p. 460. ISSN: 1999-5903. DOI: 10.3390/fi5040460. URL: http://www.mdpi.com/1999-5903/5/4/460 (cit. on pp. 36, 37).

[209] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. *TCP Selective Acknowledgment Options*. RFC 2018. http://www.rfc-editor.org/rfc/rfc2018.txt. 1996 (cit. on pp. 45, 49).

[210] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm." In: *SIGCOMM Comput. Commun. Rev.* 27.3 (July 1997), pp. 67–82. ISSN: 0146-4833. DOI: 10.1145/263932.264023 (cit. on p. 49).

[211] McAfee Labs. *Quaterly Threat Report December 2018*. https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-dec-2018.pdf. Online; accessed 2019-05-02 (cit. on pp. 138, 162).

[212] McAfee Labs. *Quaterly Threat Report June 2018*. https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-jun-2018.pdf. Online; accessed 2019-05-02 (cit. on pp. 138, 162).

[213] McAfee Labs. *Quaterly Threat Report March 2018*. https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-mar-2018.pdf. Online; accessed 2019-05-02 (cit. on pp. 138, 162).

[214] McAfee Labs. *Quaterly Threat Report September 2018*. https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-sep-2018.pdf. Online; accessed 2019-05-02 (cit. on pp. 138, 162).

[215] R. McNaughton and H. Yamada. "Regular Expressions and State Graphs for Automata." In: *IRE Transactions on Electronic Computers* EC-9.1 (Mar. 1960), pp. 39–47. ISSN: 0367-9950. DOI: 10.1109/TEC.1960.5221603 (cit. on p. 105).

[216] S. A. Mehdi, J. Khalid, and S. A. Khayam. "Revisiting Traffic Anomaly Detection Using Software Defined Networking." In: *14th International Symposium on Recent Advances in Intrusion Detection (RAID)*. Ed. by R. Sommer, D. Balzarotti, and G. Maier. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 161–180 (cit. on p. 141).

[217] J. Mirkovic, G. Prier, and P. Reiher. "Attacking DDoS at the source." In: *10th IEEE International Conference on Network Protocols, 2002. Proceedings.* Nov. 2002, pp. 312–321. DOI: 10.1109/ICNP.2002.1181418 (cit. on p. 141).

[218] J. Mirkovic, G. Prier, and P. Reiher. "Source-end DDoS defense." In: *Second IEEE International Symposium on Network Computing and Applications, 2003. NCA 2003.* Apr. 2003, pp. 171–178. DOI: 10.1109/NCA.2003.1201153 (cit. on p. 141).

[219] J. Mirkovic, A. Hussain, B. Wilson, S. Fahmy, P. Reiher, R. Thomas, W.-M. Yao, and S. Schwab. "Towards User-centric Metrics for Denial-of-service Measurement." In: *Proceedings of the 2007 Workshop on Experimental Computer Science*. ExpCS '07. San Diego, California: ACM, 2007. ISBN: 978-1-59593-751-3. DOI: 10.1145/1281700.1281708 (cit. on p. 158).

[220] J. Mirkovic, M. Robinson, and P. Reiher. "Alliance Formation for DDoS Defense." In: *Proceedings of the 2003 Workshop on New Security Paradigms*. NSPW '03. Ascona, Switzerland: ACM, 2003, pp. 11–18. ISBN: 1-58113-880-6. DOI: 10.1145/986655.986658. URL: http://doi.acm.org/10.1145/986655.986658 (cit. on p. 148).

[221] A. T. Mizrak, S. Savage, and K. Marzullo. "Detecting compromised routers via packet forwarding behavior." In: *IEEE Network* 22.2 (Mar. 2008), pp. 34–39. ISSN: 0890-8044. DOI: 10.1109/MNET.2008.4476069 (cit. on p. 141).

[222] J. Mo, R. J. La, V. Anantharam, and J. Walrand. "Analysis and Comparison of TCP Reno and Vegas." In: *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*. Vol. 3. Mar. 1999, 1556–1563 vol.3. DOI: 10.1109/INFCOM.1999.752178 (cit. on p. 48).

[223] S. Molnár, P. Megyesi, and G. Szabó. "How to validate traffic generators?" In: *2013 IEEE International Conference on Communications Workshops (ICC)*. June 2013, pp. 1340–1344. DOI: 10.1109/ICCW.2013.6649445 (cit. on p. 30).

[224] G. E. Moore. "Cramming more components onto integrated circuits." In: *Electronics* 38 (Apr. 1965) (cit. on pp. 4, 99).

[225] J. Moscola, J. Lockwood, R. P. Loui, and M. Pachos. "Implementation of a content-scanning module for an Internet firewall." In: *IEEE FCCM*. 2003 (cit. on p. 97).

[226] S. M. Mousavi and M. St-Hilaire. "Early detection of DDoS attacks against SDN controllers." In: *2015 International Conference on Computing, Networking and Communications (ICNC)*. Feb. 2015, pp. 77–81. DOI: 10.1109/ICCNC.2015.7069319 (cit. on pp. 142, 143).

[227] N. Moustafa and J. Slay. "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)." In: *Military Communications and Information Systems Conference (MilCIS), 2015*. Nov. 2015, pp. 1–6. DOI: 10.1109/MilCIS.2015.7348942 (cit. on p. 20).

[228] D. Moustis and P. Kotzanikolaou. "Evaluating Security Controls Against HTTP-based DDoS Attacks." In: *Fourth International Conference on Information Intelligence Systems and Applications (IISA)*. 2013 (cit. on pp. 145, 152, 166).

[229] D. Mutz, G. Vigna, and R. Kemmerer. "An experience developing an IDS stimulator for the black-box testing of network intrusion detection systems." In: *19th Annual Computer Security Applications Conference, 2003. Proceedings.* Dec. 2003, pp. 374–383. DOI: 10.1109/CSAC.2003.1254342 (cit. on pp. 33, 80).

[230] J. Nagle. *Congestion Control in IP/TCP Internetworks*. RFC 896. http://www.rfc-editor.org/rfc/rfc896.txt. 1984 (cit. on p. 49).

[231] T. Narten, R. Draves, and S. Krishnan. *Privacy Extensions for Stateless Address Autoconfiguration in IPv6*. RFC 4941. http://www.rfc-editor.org/rfc/rfc4941.txt. Internet Engineering Task Force, Sept. 2007 (cit. on p. 170).

[232] J. Nazario. "Distributed Denial of Service Attacks Against Independent Media and Human Rights Sites." In: *Cryptology and Information Security Series Volume 3: The Virtual Battlefield: Perspectives on Cyber Warfare* (2009). Ed. by C. Czosseck and K. Geers, pp. 163–181. DOI: 10.3233/978-1-60750-060-5-163 (cit. on p. 132).

[233] J. Nazario. "Politically Motivated Denial of Service Attacks." In: *Cryptology and Information Security Series Volume 3: The Virtual Battlefield: Perspectives on Cyber Warfare* (2009). Ed. by C. Czosseck and K. Geers (cit. on p. 132).

[234] R. Netravali, A. Sivaraman, S. Das, A. Goyal, K. Winstein, J. Mickens, and H. Balakrishnan. "Mahimahi: Accurate Record-and-Replay for HTTP." In: *2015 USENIX Annual Technical Conference (USENIX ATC 15)*. Santa Clara, CA: USENIX Association, 2015, pp. 417–429. ISBN: 978-1-931971-225 (cit. on p. 31).

[235] *New all-time peak at DE-CIX Frankfurt: 6.8 Tbps*. English. DE-CIX Management GmbH. Feb. 2016. URL: https://www.de-cix.net/en/news-events/news/new-all-time-peak-at-de-cix-frankfurt-6-8-tbps (visited on 01/08/2020) (cit. on p. 36).

[236] G. Oikonomou and J. Mirkovic. "Modeling Human Behavior for Defense Against Flash-Crowd Attacks." In: *2009 IEEE International Conference on Communications*. June 2009, pp. 1–6. DOI: 10.1109/ICC.2009.5199191 (cit. on p. 148).

[237] Q. Pan, H. Yong-feng, and Z. Pei-feng. "Reduction of traffic sampling impact on anomaly detection." In: *2012 7th International Conference on Computer Science Education (ICCSE)*. 2012 7th International Conference on Computer Science Education (ICCSE). 2012, pp. 438–443. DOI: 10.1109/ICCSE.2012.6295109 (cit. on p. 95).

[238] S. Pan, T. Morris, and U. Adhikari. "Classification of Disturbances and Cyber-Attacks in Power Systems Using Heterogeneous Time-Synchronized Data." In: *Industrial Informatics, IEEE Transactions on* 11.3 (June 2015), pp. 650–662. ISSN: 1551-3203. DOI: 10.1109/TII.2015.2420951 (cit. on p. 28).

[239] S. Pan, T. Morris, and U. Adhikari. "Developing a Hybrid Intrusion Detection System Using Data Mining for Power Systems." In: *IEEE Transactions on Smart Grid* (Nov. 2015), pp. 3104–3113. ISSN: 1949-3053. DOI: 10.1109/TSG.2015.2409775 (cit. on p. 28).

[240] S. Pan, T. Morris, and U. Adhikari. "A Specification-based Intrusion Detection Framework for Cyber-physical Environment in Electric Power System." In: *International Journal of Network Security* 17.2 (Mar. 2015), pp. 174–188 (cit. on p. 28).

[241] C. Papadopoulos, R. Lindell, J. Mehringer, A. Hussain, and R. Govindan. "COSSACK: Coordinated Suppression of Simultaneous Attacks." In: *Proceedings DARPA Information Survivability Conference and Exposition*. Vol. 2. Apr. 2003, 94–96 vol.2. DOI: 10.1109/DISCEX.2003.1194932 (cit. on p. 148).

[242] J. Park, K. Iwai, H. Tanaka, and T. Kurokawa. "Analysis of Slow Read DoS attack." In: *2014 International Symposium on Information Theory and its Applications*. Oct. 2014, pp. 60–64 (cit. on p. 138).

[243] K. Park and H. Lee. "On the effectiveness of probabilistic packet marking for IP traceback under denial of service attack." In: *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*. Vol. 1. Apr. 2001, 338–347 vol.1. DOI: 10.1109/INFCOM.2001.916716 (cit. on p. 141).

[244] K. Park and H. Lee. "On the Effectiveness of Route-based Packet Filtering for Distributed DoS Attack Prevention in Power-law Internets." In: *SIGCOMM Comput. Commun. Rev.* 31.4 (Aug. 2001), pp. 15–26. ISSN: 0146-4833. DOI: 10.1145/964723.383061. URL: http://doi.acm.org/10.1145/964723.383061 (cit. on p. 141).

[245] V. Paxson, R. Sommer, and N. Weaver. "An architecture for exploiting multi-core processors to parallelize network intrusion prevention." In: *IEEE Sarnoff Symposium*. IEEE Sarnoff Symposium. 2007, pp. 1–7. DOI: 10.1109/SARNOF.2007.4567341 (cit. on p. 96).

[246] V. Paxson. "Bro: A System for Detecting Network Intruders in Real-time." In: *USENIX Security*. San Antonio, Texas: USENIX Association, 1998 (cit. on p. 97).

[247] V. Paxson. "Bro: A System for Detecting Network Intruders in Real-Time." In: *Computer Networks*. 1999, pp. 2435–2463 (cit. on p. 92).

[248] T. Peng, C. Leckie, and K. Ramamohanarao. "Protection from distributed denial of service attacks using history-based IP filtering." In: *Communications, 2003. ICC '03. IEEE International Conference on*. Vol. 1. May 2003, 482–486 vol.1 (cit. on p. 141).

[249] B. Pfaff, B. Lantz, B. Heller, C. Barker, C. Beckmann, D. Cohn, D. T. D. Erickson, D. McDysan, D. Ward, E. Crabbe, F. Schneider, G. Gibb, G. Appenzeller, J. Tourrilhes, J. Tonsing, J. Pettit, K. Yap, L. Poutievski, L. Vicisano, M. Casado, et al. *OpenFlow Switch Specification*. ONF TS-007. Version 1.3.1 (Wire Protocol 0x04). Open Networking Foundation. Sept. 2012 (cit. on p. 169).

[250] S. Pontarelli, G. Bianchi, and S. Teofili. "Traffic-Aware Design of a High-Speed FPGA Network Intrusion Detection System." In: *IEEE Transactions On Computers*. Vol. 62. 2013, pp. 2322–2334. DOI: 10.1109/TC.2012.105 (cit. on p. 96).

[251] J. Postel. *Transmission Control Protocol*. RFC 793. Updated by RFCs 1122, 3168, 6093, 6528. Internet Engineering Task Force, Sept. 1981 (cit. on p. 158).

[252] R. Pries, Z. Magyari, and P. Tran-Gia. "An HTTP web traffic model based on the top one million visited web pages." In: *Proceedings of the 8th Euro-NF Conference on Next Generation Internet NGI 2012*. June 2012, pp. 133–139. DOI: 10.1109/NGI.2012.6252145 (cit. on pp. 30, 66, 67, 217).

[253] N. J. Puketza, K. Zhang, M. Chung, B. Mukherjee, and R. A. Olsson. "A methodology for testing intrusion detection systems." In: *IEEE Transactions on Software Engineering* 22.10 (Oct. 1996), pp. 719–729. ISSN: 0098-5589. DOI: 10.1109/32.544350 (cit. on p. 33).

[254] M. O. Rabin and D. Scott. "Finite Automata and Their Decision Problems." In: *IBM Journal of Research and Development* 3.2 (Apr. 1959), pp. 114–125. ISSN: 0018-8646. DOI: 10.1147/rd.32.0114 (cit. on p. 106).

[255] S. Ranjan, R. Swaminathan, M. Uysal, A. Nucci, and E. Knightly. "DDoS-Shield: DDoS-Resilient Scheduling to Counter Application Layer Attacks." In: *IEEE/ACM Transactions on Networking* 17.1 (Feb. 2009), pp. 26–39. ISSN: 1063-6692. DOI: 10.1109/TNET.2008.926503 (cit. on p. 147).

[256] A. Reed and J. Aikat. "Modeling, identifying, and simulating Dynamic Adaptive Streaming over HTTP." In: *2013 21st IEEE International Conference on Network Protocols (ICNP)*. Oct. 2013, pp. 1–2. DOI: 10.1109/ICNP.2013.6733626 (cit. on pp. 30, 70, 219).

[257] K. Rieck and P. Laskov. "Detecting unknown network attacks using language models." In: *Detection of Intrusions and Malware & Vulnerability Assessment* (2006), pp. 74–90. URL: http://link.springer.com/chapter/10.1007/11790754_5 (cit. on p. 91).

[258] M. Ring and S. Wunderlich. *Technical Report CIDDS-002 data set*. Tech. rep. 2017 (cit. on p. 19).

[259] M. Ring, S. Wunderlich, and D. Grüdl. *Technical Report CIDDS-001 data set*. Tech. rep. 2017 (cit. on p. 19).

[260] M. Ring, S. Wunderlich, D. Grüdl, D. Landes, and A. Hotho. "Flow-based benchmark data sets for intrusion detection." In: *Journal of Information Warfare*. 2016 (cit. on p. 19).

[261] M. Ring, S. Wunderlich, D. Grüdl, D. Landes, and A. Hotho. "Flow-based benchmark data sets for intrusion detection." In: *Proceedings of the 16th European Conference on Cyber Warfare and Security*. 2017 (cit. on p. 19).

[262] M. Ring, S. Wunderlich, D. Scheuring, D. Landes, and A. Hotho. "A Survey of Network-based Intrusion Detection Data Sets." In: *Computer & Security* (2019). eprint: 1903.02460 (cit. on pp. 20, 29).

[263] M. Roesch. "Snort - Lightweight Intrusion Detection for Networks." In: *Proceedings of the 13th USENIX Conference on System Administration*. LISA '99. Seattle, Washington: USENIX Association, 1999. URL: http://dl.acm.org/citation.cfm?id=1039834.1039864 (cit. on pp. 92, 97).

[264] C. Rossow. "Amplification Hell: Revisiting Network Protocols for DDoS Abuse." In: *Proceedings of the 2014 Network and Distributed System Security (NDSS) Symposium*. Feb. 2014. URL: http://www.christian-rossow.de/publications/amplification-ndss2014.pdf (cit. on p. 138).

[265] J. Ryan, M.-J. Lin, and R. Miikkulainen. "Intrusion Detection With Neural Networks." In: *Advances in Neural Information Processing Systems*. Ed. by M. I. Jordan, M. J. Kearns, and S. A. Solla. Cambridge, MA: MIT Press, 1998, pp. 943–949. URL: http://nn.cs.utexas.edu/?ryan:nips97 (cit. on p. 91).

[266] S. Saad, I. Traore, A. Ghorbani, B. Sayed, D. Zhao, W. Lu, J. Felix, and P. Hakimian. "Detecting P2P botnets through network behavior analysis and machine learning." In: *2011 Ninth Annual International Conference on Privacy, Security and Trust*. July 2011, pp. 174–180. DOI: 10.1109/PST.2011.5971980 (cit. on p. 27).

[267] Sandvine. *2015 Global Internet Phenomena: Asia-Pacific and Europe*. https://www.sandvine.com/hubfs/downloads/archive/2015-global-internet-phenomena-report-apac-and-europe.pdf. Online; accessed 2018-02-07 (cit. on pp. 66, 68, 209).

[268] Sandvine. *2016 Global Internet Phenomena: Africa, Asia-Pacific and Middle East*. https://www.sandvine.com/hubfs/downloads/archive/2016-global-internet-phenomena-apac-mea.pdf. Online; accessed 2018-02-07 (cit. on pp. 66, 209).

[269] Sandvine. *2016 Global Internet Phenomena: Latin America and North America*. https://www.sandvine.com/hubfs/downloads/archive/2016-global-internet-phenomena-report-latin-america-and-north-america.pdf. Online; accessed 2018-02-07 (cit. on pp. 66, 209).

[270] B. Sangster, T. J. O'Connor, T. Cook, R. Fanelli, E. Dean, W. J. Adams, C. Morrell, and G. Conti. "Toward Instrumenting Network Warfare Competitions to Generate Labeled Datasets." In: *Proceedings of the 2nd Conference on Cyber Security Experimentation and Test*. CSET'09. Montreal, Canada: USENIX Association,

2009, pp. 9–9. URL: http://dl.acm.org/citation.cfm?id=1855481.1855490 (cit. on p. 23).

[271] J. Santanna, R. van Rijswijk-Deij, R. Hofstede, A. Sperotto, M. Wierbosch, L. Zambenedetti Granville, and A. Pras. "Booters - An analysis of DDoS-as-a-service attacks." In: *IFIP/IEEE International Symposium on Integrated Network Management (IM)*. May 2015, pp. 243–251. DOI: 10.1109/INM.2015.7140298 (cit. on pp. 24, 185).

[272] A. Santos da Silva, J. A. Wickboldt, L. Z. Granville, and A. Schaeffer-Filho. "ATLANTIC: A framework for anomaly traffic detection, classification, and mitigation in SDN." In: *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*. Apr. 2016, pp. 27–35. DOI: 10.1109/NOMS.2016.7502793 (cit. on pp. 142, 145, 152).

[273] D. Sattar, A. Matrawy, and O. Adeojo. "Adaptive Bubble Burst (ABB): Mitigating DDoS attacks in Software-Defined Networks." In: *2016 17th International Telecommunications Network Strategy and Planning Symposium (Networks)*. Sept. 2016, pp. 50–55. DOI: 10.1109/NETWKS.2016.7751152 (cit. on p. 147).

[274] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. "Practical Network Support for IP Traceback." In: *SIGCOMM Comput. Commun. Rev.* 30.4 (Aug. 2000), pp. 295–306. ISSN: 0146-4833. DOI: 10.1145/347057.347560. URL: http://doi.acm.org/10.1145/347057.347560 (cit. on p. 146).

[275] F. Schneider, J. Wallerich, and A. Feldmann. "Packet Capture in 10-Gigabit Ethernet Environments Using Contemporary Commodity Hardware." In: *Passive and Active Network Measurement*. Ed. by S. Uhlig, K. Papagiannaki, and O. Bonaventure. Vol. 4427. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2007, pp. 207–217. ISBN: 978-3-540-71616-7. DOI: 10.1007/978-3-540-71617-4_21. URL: http://dx.doi.org/10.1007/978-3-540-71617-4_21 (cit. on p. 96).

[276] H. Schulze and K. Mochalski. *ipoque internet study 2008/2009*. ipoque. 2009. URL: http://portal.ipoque.com/downloads/index/study (cit. on p. 68).

[277] M. Scott. *A Wire-speed Packet Classification and Capture Module for NetFPGA*. Tech. rep. 2010 (cit. on p. 110).

[278] T. Seals. *DDoS Attacks Get Bigger, Smarter and More Diverse*. https://threatpost.com/ddos-attacks-get-bigger-smarter-and-more-diverse/134028/. Online; accessed 2019-04-24 (cit. on p. 132).

[279] S. Shanklin and G. Lathem. *Parallel intrusion detection sensors with load balancing for high speed networks*. US Patent 6,578,147. 2003. URL: http://www.google.com/patents/US6578147 (cit. on p. 96).

[280] I. Sharafaldin, A. Gharib, A. H. Lashkari, and A. A. Ghorbani. "Towards a Reliable Intrusion Detection Benchmark Dataset." In: *River Journal* (2017), pp. 177–200 (cit. on pp. 23, 26).

[281] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani. "Toward developing a systematic approach to generate benchmark datasets for intrusion detection." In: *Computers & Security* 31.3 (2012), pp. 357–374. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2011.12.012. URL: http://www.sciencedirect.com/science/article/pii/S0167404811001672 (cit. on p. 20).

[282] R. Sidhu and V. K. Prasanna. "Fast Regular Expression Matching Using FPGAs." In: *The 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'01)*. 2001 (cit. on p. 97).

[283] *Slowloris HTTP DoS*. http://ha.ckers.org/slowloris/. Online; accessed: 2020-01-08 via web.archive.org. June 2009 (cit. on p. 137).

[284] H. Song and J. W. Lockwood. "Efficient Packet Classification for Network Intrusion Detection Using FPGA." In: *Proceedings of the 2005 ACM/SIGDA 13th International Symposium on Field-programmable Gate Arrays*. FPGA '05. New York, NY, USA: ACM, 2005, pp. 238–245. ISBN: 1-59593-029-9. DOI: 10.1145/1046192.1046223. URL: http://doi.acm.org/10.1145/1046192.1046223 (cit. on p. 96).

[285] H. Song, T. Sproull, M. Attig, and J. Lockwood. "Snort offloader: a reconfigurable hardware NIDS filter." In: *International Conference on Field Programmable Logic and Applications*. 2005 (cit. on p. 97).

[286] A. Sperotto, R. Sadre, F. van Vliet, and A. Pras. "A Labeled Data Set for Flow-Based Intrusion Detection." In: *IP Operations and Management*. Ed. by G. Nunzi, C. Scoglio, and X. Li. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 39–50. ISBN: 978-3-642-04968-2 (cit. on p. 24).

[287] Spirent. *Spirent TestCenter Software*. 2019. URL: https://www.spirent.com/products/testcenter/platforms/software (visited on 05/13/2019) (cit. on p. 144).

[288] P. Srisuresh and K. Egevang. *Traditional IP Network Address Translator (Traditional NAT)*. RFC 3022. RFC. http://www.rfc-editor.org/rfc/rfc3022.txt. 2001 (cit. on p. 173).

[289] M. Srivatsa, A. Iyengar, J. Yin, and L. Liu. "Mitigating Application-level Denial of Service Attacks on Web Servers: A Client-transparent Approach." In: *ACM Transactions on the Web (TWEB)* 2.3 (July 2008), 15:1–15:49. ISSN: 1559-1131. DOI: 10.1145/1377488.1377489. URL: http://doi.acm.org/10.1145/1377488.1377489 (cit. on p. 148).

[290] D. Steahle, K. Leibnitz, and P. Tran-Gia. "Source Traffic Modeling of Wireless Applications." In: *AEU - International Journal of Electronics and Communications* Volume 55 Issue 1 (2001), 27–36 (cit. on p. 30).

[291] Suricata. *Deprecation Policy*. URL: https://suricata-ids.org/about/deprecation-policy/ (visited on 05/31/2019) (cit. on p. 96).

[292] R. Susitaival and S. Aalto. "Modelling the Population Dynamics and the File Availability in a Bittorrent-like P2P System with Decreasing Peer Arrival Rate." In: *Proceedings of the First International Conference, and Proceedings of the Third International Conference on New Trends in Network Architectures and Services Conference on Self-Organising Systems*. IWSOS'06/EuroNGI'06. Passau, Germany: Springer-Verlag, 2006, pp. 34–48. ISBN: 3-540-37658-5, 978-3-540-37658-3. DOI: 10.1007/11822035_5. URL: http://dx.doi.org/10.1007/11822035_5 (cit. on p. 30).

[293] R. Swami, M. Dave, and V. Ranga. "Software-defined Networking-based DDoS Defense Mechanisms." In: *ACM Computing Surveys (CSUR)* 52.2 (Apr. 2019), 28:1–28:36. ISSN: 0360-0300. DOI: 10.1145/3301614. URL: http://doi.acm.org/10.1145/3301614 (cit. on pp. 132, 133, 140).

[294] Y. Tao and S. Yu. "DDoS Attack Detection at Local Area Networks Using Information Theoretical Metrics." In: *12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. July 2013, pp. 233–240. DOI: 10.1109/TrustCom.2013.32 (cit. on p. 142).

[295] M. Tavallaee, E. Bagheri, W. Lu, and A. Ghorbani. "A detailed analysis of the KDD CUP 99 data set." In: *Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on*. July 2009, pp. 1–6. DOI: 10.1109/CISDA.2009.5356528 (cit. on pp. 22, 23).

[296] S. Tayama and H. Tanaka. "Analysis of Slow Read DoS Attack and Communication Environment." In: *Mobile and Wireless Technologies 2017: ICMWT 2017*. Ed. by K. J. Kim and N. Joukov. Springer Singapore, 2018, pp. 350–359. ISBN: 978-981-10-5281-1 (cit. on p. 138).

[297] M. K. for The Hacker News. https://thehackernews.com/2018/03/ddos-attack-memcached.html. Online; accessed 2019-03-24 (cit. on pp. 4, 131, 132).

[298] The Hacker's Choice via the WaybackMachine. http://web.archive.org/web/20160311191721/https://www.thc.org/thc-ssl-dos/. Wayback accessed 2019-05-02, archive from 2016-03-11 (cit. on p. 137).

[299] The Linux Foundation. *NetEm*. http://www.linuxfoundation.org/collaborate/workgroups/networking/netem. (Visited on 05/04/2016) (cit. on p. 32).

[300] *The Suricata Open Source IDS/IPS/NSM Engine*. URL: http://suricata-ids.org (visited on 03/27/2015) (cit. on pp. 92, 96).

[301] *The Zeek Network Security Monitor*. URL: http://www.zeek.org (visited on 05/31/2019) (cit. on p. 96).

[302] K. Thompson. "Programming Techniques: Regular Expression Search Algorithm." In: *Communications of the ACM* 11.6 (June 1968), pp. 419–422. ISSN: 0001-0782. DOI: 10.1145/363347.363387. URL: http://doi.acm.org/10.1145/363347.363387 (cit. on p. 105).

[303] N. Tripathi, N. Hubballi, and Y. Singh. "How Secure are Web Servers? An Empirical Study of Slow HTTP DoS Attacks and Detection." In: *2016 11th International Conference on Availability, Reliability and Security (ARES)*. Aug. 2016, pp. 454–463. DOI: 10.1109/ARES.2016.20 (cit. on pp. 145, 152).

[304] M. J. M. Turcotte, A. D. Kent, and C. Hash. *Unified Host and Network Data Set*. Tech. rep. 2017 (cit. on p. 26).

[305] M. Vallentin, R. Sommer, J. Lee, C. Leres, V. Paxson, and B. Tierney. "The NIDS Cluster: Scalable, Stateful Network Intrusion Detection on Commodity Hardware." In: *Proceedings of the 10th International Symposium on Recent Advances in Intrusion Detection (RAID)*. 2007, pp. 107–126. ISBN: 9783540743194. DOI: 10.1007/978-3-540-74320-0_6 (cit. on p. 96).

[306] M. Vallentin, R. Sommer, J. Lee, C. Leres, V. Paxson, and B. Tierney. "The NIDS Cluster: Scalable, Stateful Network Intrusion Detection on Commodity Hardware." In: *RAID*. Gold Goast, Australia, 2007. ISBN: 978-3-540-74319-4 (cit. on p. 97).

[307] G. Vasiliadis, S. Antonatos, M. Polychronakis, E. P. Markatos, and S. Ioannidis. "Gnort: High Performance Network Intrusion Detection Using Graphics Processors." In: *11th International Symposium on Recent Advances in Intrusion Detection (RAID)*. Ed. by R. Lippmann, E. Kirda, and A. Trachtenberg. Lecture Notes in Computer Science 5230. Springer Berlin Heidelberg, 2008, pp. 116–134. ISBN: 978-3-540-87402-7, 978-3-540-87403-4. URL: http://link.springer.com/chapter/10.

1007/978-3-540-87403-4%5C_7 (cit. on pp. 95, 99, 100, 126, 207).

[308] E. Vasilomanolakis, C. G. Cordero, N. Milanov, and M. Mühlhäuser. "Towards the creation of synthetic, yet realistic, intrusion detection datasets." In: *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*. Apr. 2016, pp. 1209–1214. DOI: 10.1109/NOMS.2016.7502989 (cit. on p. 31).

[309] A. Verma and V. Ranga. "Statistical analysis of CIDDS-001 dataset for Network Intrusion Detection Systems using Distance-based Machine Learning." In: *Procedia Computer Science* 125 (2018). The 6th International Conference on Smart Computing and Communications, pp. 709–716. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2017.12.091. URL: http://www.sciencedirect.com/science/article/pii/S1877050917328594 (cit. on p. 19).

[310] F. Veysset, O. Courtay, and O. Heen. *New tool and technique for remote operating system fingerprinting*. Tech. rep. 2002 (cit. on p. 72).

[311] N. Vicari and S. Koehler. *Measuring Internet User Traffic Behavior Dependent on Access Speed*. Technial Report No. 238. Institute of Computer Science, University of Würzburg, 1999 (cit. on p. 30).

[312] E. K. Viegas, A. O. Santin, and L. S. Oliveira. "Toward a reliable anomaly-based intrusion detection in real-world environments." In: *Computer Networks* 127 (2017), pp. 200–216. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2017.08.013. URL: http://www.sciencedirect.com/science/article/pii/S1389128617303225 (cit. on p. 22).

[313] J.-B. Voron, C. Démoulins, and F. Kordon. "Adaptable Intrusion Detection Systems Dedicated to Concurrent Programs: A Petri Net-Based Approach." In: *10th International Conference on Application of Concurrency to System Design*. Washington, DC, USA: IEEE Computer Society, 2010, pp. 57–66 (cit. on p. 166).

[314] Vuze Bittorrent Client Wiki. *Good Settings*. URL: https://wiki.vuze.com/w/Good_settings (cit. on p. 218).

[315] W3 Web Technology Surveys. *Usage statistics of Gzip Compression for websites*. https://w3techs.com/technologies/details/ce-gzipcompression/all/all. Online; accessed 2019-08-25 (cit. on pp. 30, 67).

[316] M. Walfish, M. Vutukuru, H. Balakrishnan, D. Karger, D. Karger, and S. Shenker. "DDoS Defense by Offense." In: *Proceedings of the 2006 conference on Applications, technologies,*

architectures, and protocols for computer communications (SIGCOMM) 36.4 (Aug. 2006), pp. 303–314. ISSN: 0146-4833. DOI: 10.1145/1151659.1159948. URL: http://doi.acm.org/10.1145/1151659.1159948 (cit. on p. 148).

[317] B. Wang, Y. Zheng, W. Lou, and Y. T. Hou. "DDoS attack protection in the era of cloud computing and Software-Defined Networking." In: *Computer Networks* 81 (2015), pp. 308–319. ISSN: 1389-1286. DOI: http://doi.org/10.1016/j.comnet.2015.02.026 (cit. on p. 147).

[318] H. Wang, C. Jin, and K. G. Shin. "Defense Against Spoofed IP Traffic Using Hop-Count Filtering." In: *IEEE/ACM Transactions on Networking* 15.1 (Feb. 2007), pp. 40–53. ISSN: 1063-6692. DOI: 10.1109/TNET.2006.890133 (cit. on p. 144).

[319] K. Wang, J. Parekh, and S. Stolfo. "Anagram: A content anomaly detector resistant to mimicry attack." In: *Proceedings of the 9th International Symposium on Recent Advances in Intrusion Detection (RAID)*. 2006. URL: http://link.springer.com/chapter/10.1007/11856214_12 (cit. on p. 91).

[320] N. Weaver, V. Paxson, and J. M. Gonzalez. "The Shunt: An FPGA-based Accelerator for Network Intrusion Prevention." In: *ACM/SIGDA FPGA*. Monterey, California, USA, 2007. ISBN: 978-1-59593-600-4 (cit. on pp. 95, 97).

[321] D. Wei, P. Cao, and S. Low. *Time for a TCP Benchmark Suite?* Tech. rep. Caltech, 2005 (cit. on pp. 32, 40, 44).

[322] W. Wei, F. Chen, Y. Xia, and G. Jin. "A rank correlation based detection against distributed reflection DoS attacks." In: *IEEE Communications Letters* (2013) (cit. on p. 143).

[323] M. Weigle, P. Sharma, and J. Freemen. "Performance of Competing High-Speed TCP Flows." In: *NETWORKING 2006*. Ed. by F. Boavida, T. Plagemann, B. Stiller, C. Westphal, and E. Monteiro. Berlin, Germany: Springer, 2006, 476–487 (cit. on pp. 37, 38, 44).

[324] G. C. Wilshusen and D. C. Trimble. *GAO-12-926T Challenges in Securing the Electricity Grid*. Tech. rep. United States Government Accountability Office. July 2012 (cit. on p. 36).

[325] C. Wressnegger, G. Schwenk, D. Arp, and K. Rieck. "A Close Look on n-Grams in Intrusion Detection: Anomaly Detection vs. Classification." In: *AISec '13 Proceedings of the 2013 ACM workshop on Artificial intelligence and security*. ACM New York, NY, USA ©2013, 2013, pp. 67–76. ISBN: 9781450324885 (cit. on p. 91).

[326] Y. Wu, H. Tseng, W. Yang, and R. Jan. "DDoS Detection and Traceback with Decision Tree and Grey Relational Analysis." In: *2009 Third International Conference on Multimedia and Ubiquitous Engineering*. June 2009, pp. 306–314. DOI: 10.1109/MUE.2009.60 (cit. on p. 146).

[327] J. Xu and C. R. Shelton. "Intrusion Detection using Continuous Time Bayesian Networks." In: *Journal of Artificial Intelligence Research* (2014). DOI: 10.1613/jair.3050 (cit. on p. 91).

[328] L. Xu, K. Harfoush, and I. Rhee. "Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks." In: *Proceedings of the INFOCOM 2004. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies*. 2004 (cit. on p. 46).

[329] L. Xu, K. Harfoush, and I. Rhee. "Experimental Evaluation of TCP Performance over 10Gb/s Passive Optical Networks (XG-PON)." In: *Proceedings of the Globecom 2014 - Symposium on Selected Areas in Communications: GC14 SAC Access Networks and Systems*. 2014 (cit. on p. 39).

[330] Xu, K. *Performance Modeling of BitTorrent Peer-to-Peer File Sharing Networks*. Tech. rep. 2013 (cit. on p. 30).

[331] A. Yaar, A. Perrig, and D. Song. "Pi: a path identification mechanism to defend against DDoS attacks." In: *2003 Symposium on Security and Privacy, 2003*. May 2003, pp. 93–107. DOI: 10.1109/SECPRI.2003.1199330 (cit. on p. 144).

[332] N. Yamagaki, R. Sidhu, and S. Kamiya. "High-speed regular expression matching engine using multi-character NFA." In: *International Conference on Field Programmable Logic and Applications*. 2008. DOI: 10.1109/FPL.2008.4629920 (cit. on pp. 97, 113).

[333] Y.-H. E. Yang, W. Jiang, and V. K. Prasanna. "Compact Architecture for High-throughput Regular Expression Matching on FPGA." In: *ACM/IEEE ANCS*. 2008. ISBN: 978-1-60558-346-4 (cit. on pp. 97, 113).

[334] D. K. Y. Yau and J. C. S. Lui. "Defending against distributed denial-of-service attacks with max-min fair server-centric router throttles." In: *IEEE 2002 Tenth IEEE International Workshop on Quality of Service (Cat. No.02EX564)*. May 2002, pp. 35–44. DOI: 10.1109/IWQoS.2002.1006572 (cit. on p. 147).

[335] N. Ye, S. Emran, Q. Chen, and S. Vilbert. "Multivariate statistical analysis of audit trails for host-based intrusion detection." In: *IEEE Transactions on Computers* 51.7 (2002), pp. 810–820. ISSN: 0018-9340. DOI: 10.1109/TC.2002.1017701 (cit. on p. 90).

[336] N. Ye. "A Markov Chain Model of Temporal Behavior for Anomaly Detection." In: *In Proceedings of the 2000 IEEE Workshop on Information Assurance and Security*. 2000, pp. 171–174 (cit. on p. 91).

[337] D.-y. Yeung and Y. Ding. "Host-based intrusion detection using dynamic and static behavioral models." In: *Pattern Recognition* 36 (2003), pp. 229–243 (cit. on p. 91).

[338] S. T. Zargar, J. Joshi, and D. Tipper. "A Survey of Defense Mechanisms Against Distributed Denial of Service (DDoS) Flooding Attacks." In: *IEEE Communications Surveys Tutorials* 15.4 (Apr. 2013), pp. 2046–2069. ISSN: 1553-877X. DOI: 10.1109/SURV.2013.031413.00127 (cit. on p. 139).

[339] B. Zdrnja. *Slowloris and Iranian DDoS attacks*. https://isc.sans.edu/diary/6622. Online; accessed: 2020-01-08. June 2009 (cit. on p. 137).

[340] G. Zhang, S. Ehlert, T. Magedanz, and D. Sisalem. "Denial of Service Attack and Prevention on SIP VoIP Infrastructures Using DNS Flooding." In: *Proceedings of the 1st International Conference on Principles, Systems and Applications of IP Telecommunications*. IPTComm '07. New York City, New York: ACM, 2007, pp. 57–66. ISBN: 978-1-60558-006-7. DOI: 10.1145/1326304.1326314. URL: http://doi.acm.org/10.1145/1326304.1326314 (cit. on p. 146).

[341] J. Zhang, Z. Qin, L. Ou, P. Jiang, J. Liu, and A. X. Liu. "An advanced entropy-based DDOS detection scheme." In: *2010 International Conference on Information, Networking and Automation (ICINA)*. Vol. 2. Oct. 2010, pp. 67–71. DOI: 10.1109/ICINA.2010.5636786 (cit. on pp. 142, 143).

[342] J. Zheng, Q. Li, G. Gu, J. Cao, D. K. Y. Yau, and J. Wu. "Realtime DDoS Defense Using COTS SDN Switches via Adaptive Correlation Analysis." In: *IEEE Transactions on Information Forensics and Security* 13.7 (July 2018), pp. 1838–1853. ISSN: 1556-6013. DOI: 10.1109/TIFS.2018.2805600 (cit. on pp. 142, 152).

[343] R. Zuech, T. Khoshgoftaar, N. Seliya, M. Najafabadi, and C. Kemp. *A New Intrusion Detection Benchmarking System*. 2015. URL: https://www.aaai.org/ocs/index.php/FLAIRS/FLAIRS15/paper/view/10368 (cit. on pp. 22, 27).