Eberhard Karls Universität Tübingen
Faculty of Science
Department of Computer Science
Chair of Communication Networks

Master Thesis
Computer Science

Modeling and Simulation of the Performance Impact of Network Security Mechanisms on Network Traffic

Nils Thomas Frank Lohmiller

03.07.2022

Reviewers

Prof. Dr. habil. Michael Menth
Department of Computer Science
Chair of Communication Networks
Eberhard Karls Universität Tübingen

Prof. Dr. rer. nat. Tobias Heer
Faculty Computer Science and Engineering
University of Applied Sciences Esslingen

Supervisor

Lukas Wüsteney, M.Sc.

Nils Thomas Frank Lohmiller:

Modeling and Simulation of the Performance Impact of Network Security Mechanisms on Network Traffic
Master Thesis Computer Science
Eberhard Karls Universität Tübingen
Thesis period: 03.12.2021 - 03.07.2022

Erklärung

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbstständig und nur mit den angegebenen Hilfsmitteln angefertigt habe und dass alle Stellen, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen sind, durch Angaben von Quellen als Entlehnung kenntlich gemacht worden sind. Diese Masterarbeit wurde in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt.

Tübingen, den 03.07.2022

(Nils Thomas Frank Lohmiller)

Contents

Abstract

Digitalization in the context of Industry 4.0 poses new challenges for industrial networks. Part of industrial digitalization is the connectivity between multiple networks to improve inter-machine communication. One challenge that arises in this context is the processing of real-time critical data across multiple networks. Until now, the real-time capability was only available within single networks.

Simultaneously, companies increase their network security to protect industrial networks from attackers. Firewalls are a fundamental part of network security. These firewalls serve as access control between the different networks. Related work shows that firewalls are not fully capable of real-time packet processing. We want to analyze to what extent this is true and predict the firewall behavior to enable industrial and real-time traffic on firewalls.

In our thesis, we create a model for software firewalls that predicts the latency, jitter, and packet loss behavior. We implement and test our model for three different software firewalls. From our prediction results, we evaluate whether our model can predict the behavior of the firewalls and can support the selection of appropriate firewalls for industrial networks at the planning stage.

1 Introduction

The digitalization of industry toward intelligent and flexible production creates new challenges for networks. Additional services integrate into existing networks, or the complexity increases due to the higher degree of automation. One example of the increasing challenges in industrial networks are sensor-to-cloud systems. For intelligent and flexible production, sensor data is essential. The sensor data provides information, for example, on the production status, the condition of machines, and the production quality. Intelligent and flexible production is only possible by sending real-time critical data reliably across multiple zones. In industrial networks, Ethernet is becoming more and more common to enable the networking of many different industrial devices. The introduction of Ethernet in industrial networks transfers the real-time critical traffic requirements of fieldbus to Ethernet. Since standard Ethernet does not meet the requirements of real-time critical traffic, the Time Sensitive Networking Task Group developed Time-Sensitive Networking (TSN). TSN is a collection of Ethernet standards to reduce latency and variance in latency (jitter) of time-critical network traffic. The reduction of latency and jitter enables the time-deterministic forwarding of packets. An industrial network contains many devices with different security requirements. We group the devices into multiple zones, e.g., Virtual Local Area Networks (VLANs), according to their network security requirements. At the edge of the zones, there are firewalls to guarantee access control. With the help of policies, the firewall decides which packets can enter the zone and which cannot.

We discuss the transmission of real-time data across zones using sensor-to-cloud as an example. As soon as we want to send sensor data to the cloud and use it for our production, we have to make sure that it is sent reliably in real time. In the case of sensor-to-cloud systems, we need to send this data across multiple zones. Until now, we can send real-time critical data only within one zone. The effects of firewalls on network traffic performance are not fully known. Additionally, there is no firewall that we can use in a TSN network [1]. Wüsteney et al. [2] discuss in their work that firewalls are not fully capable of real-time packet processing. It requires more information on how firewalls affect the performance of network traffic to enable real-time traffic across firewalls.

Our goal is to model and simulate the behavior of software firewalls. With our model, we can make statements about how a firewall behaves with specific network configurations and compare it to new network requirements.

We partition this thesis into eight chapters. In Chapter 2, we describe the motivation for this work in more detail. Furthermore, we define the goals of our thesis. Next, in Chapter 3, we explain background information that is important for understanding this thesis. Chapter 4, provides us with a review of related work and shows which processes we can adopt. In the design chapter, Chapter 5, we

present our latency and packet loss measurement setup, the system model for our firewall behavior modeling, and which influencing factors we consider. Afterward, in Chapter 6, we describe how we implement our modeling and the parameters that influence it. Chapter 7 contains our measurement and modeling results for all measured Device Under Test (DUT). In the final Chapter 8, we summarize our work and give an outlook on future work.

2 Motivation

In this chapter, we first describe challenges that originate from industrial network digitalization. Based on this, we show what goal we are pursuing with our work. Finally, we discuss the firewall type we use in this thesis.

2.1 Digitalization of Industrial Networks

The industry is becoming more and more reliant on digitalization. One goal of digitalization in industrial networks is the connectivity of different networks with each other and thus improving machine-to-machine communication [3]. Due to the growing industry digitalization, it is necessary to adapt existing and new networks to the increasing challenges. The digitalization of industrial networks requires reliable real-time communication. Information about how network security devices affect the latency, variation in latency (jitter), and packet loss of network traffic are necessary to enable reliable real-time traffic across network boundaries. Industrial networks and real-time traffic have specific requirements regarding latency, jitter, and packet loss. In industrial networks, packet loss is prohibited because the systems usually have to function around the clock without failure. Real-time traffic also allows no packet loss and expects latencies of less than 20 ms for cyclic traffic, see Section 3.9.1. So far, it was only necessary to establish real-time capability within one network. In zones and conduit networks, the real-time communication across different networks creates new problems, see Section 3.9.2. In a zones and conduit network, devices are grouped into zones, e.g., as a VLAN. These networks can only communicate via conduits. At these conduits, there are, for example, firewalls for access control. Until now, the research focused only on the bandwidth of firewalls.

2.2 Goals

Firewalls serve as a fundamental tool to increase the security of a network. We use firewalls in various network security architectures, refer to Section 3.9. So far, there is no method to predict the behavior of network security devices in terms of latency, jitter, and packet loss. The goal of our work is the creation of a model that represents the latency, jitter, and packet loss behavior of a software firewall in industrial networks. Important for our model is the consideration of industry networks. In industrial networks, packet loss is not permissible since the systems have to function around the clock without failure. We investigate anomalies in the behavior of software firewalls with regard to their latency and packet loss. We explain these anomalies and, if necessary, make suggestions for optimization. For new devices, we create our model automatically. Using our model, we discuss which

firewalls are suitable for which type of real-time traffic. The users can use our model to select the appropriate software firewall for their industrial network in advance.

2.3 Software Firewalls

Setting up an industrial network is cost-intensive. The industrial network of a company consists of several subnetworks. The subnetworks are divided, both logically and physically. Each subnetwork has specific security, packet loss, latency, and jitter requirements. High investments in additional devices are necessary to ensure the security of the subnetworks. Companies save money by using software firewalls instead of hardware firewalls wherever possible. Software firewalls do not rely on special hardware. Hence, we can use commodity hardware to build a software firewall. In addition, a hardware-independent software firewall comes without firewall rule restriction.

Hardware firewalls, on the other hand, with an Application-Specific Integrated Circuit (ASIC) are limited by the capabilities of the ASIC. Thus, there are firewall rule restrictions on hardware firewalls. A hardware firewall is specially built to support the firewall software running on it [4]. Accordingly, a hardware firewall is more expensive than a software firewall, and network designers use them when a less expensive software firewall is not an option.

Therefore, we focus on software firewalls, as they are a prerequisite for low-cost industrial networks, are frequently used, and have fewer hardware limitations.

3 Background

In our thesis, we model the performance impact of firewalls on network traffic. Therefore, it is essential to know some basics about firewalls. Second, we explain Linux kernel space networking since iptables is based on it. Third, we show how user space networking works under Linux. Fourth, we discuss the influence of temperature on the performance of computers. Afterward, we explain the letter-value plot. Next, we describe curve fitting and resulting problems. After that, we discuss the simulation framework *Objective Modular Network Testbed in C++ (OMNeT++)*. Finally, we provide a summary of common network architectures.

3.1 Firewalls

Firewalls are a fundamental component of a secure network [5]. Firewalls are located either at the edge of a network or directly on a host system. Both types of firewalls have the same goal, a method to control network access. According to Noonan et al. [5], firewalls, in their simplest form, are pure access control enforcement points. A firewall separates a protected network area from an unprotected network area. Firewalls only allow data to pass through that meets the restrictions. The firewall discards other data packets. The firewall filters prevent unwanted and possible malicious traffic from accessing the network or host system. The firewall filters not only incoming traffic but also outgoing traffic. Firewalls operate in various ways to guarantee network security. Below we describe different types of firewalls.

Packet Inspection Firewall

The packet inspection firewall uses a set of rules to make decisions about forwarding or blocking traffic. The firewall decides to forward or reject a packet based on the Internet Protocol (IP) address, port number, and protocol type [6]. The network administrator can set the decision rules for this.

For filtering packets, there are stateful rules and stateless rules. A stateless filter operates mainly on the data link layer, network layer, and transport layer of the ISO/OSI model. For the filter decision, stateless rules use only information that the packet contains. Stateless rules ignore information about which connection the packet belongs to or which connection state the packet represents. Table 3.1 shows an example entry for a stateless firewall rule. In our example, the packet is selected based on the source IP and port and the protocol type. The firewall compares each packet with this rule. The action specifies what happens to the matching packet. *Accept* indicates that the firewall forwards the packet. *Drop* means that the firewall rejects the packet.

Src. IP	Src. port	Dst. IP	Dst. port	Protocol	Flags	Action
192.168.1.2	1234	anywhere	-	UDP	-	ACCEPT

Table 3.1: Forward chain stateless rule example (Src. and Dst. are acronyms for Source and Destination); Internet Protocol (IP); User Datagram Protocol (UDP)

In contrast, stateful rules use additional information about the connection for their decision. The firewall stores connections in a connection table. With this type of filtering, the firewall can detect whether the packet belongs to an active connection and allow or reject the packet based on the session status. Table 3.2 shows an example of an entry from a connection table. The entry contains information about the packet, the connection state, and a timeout. The connection state indicates the status of the connection. In our example, the state is *ESTABLISHED*. This means that the connection is already existing. The timeout specifies the time in seconds, after which the firewall deletes the entry. If the firewall processes a packet of the corresponding connection within the timeout, the timeout is reset. For existing Transmission Control Protocol (TCP) connections, the default timeout is five days [7]. That is the reason for a timeout of 431,999 seconds in our table.

Src. IP	Src. port	Dst. IP	Dst. port	Connection state	Timeout
192.168.10.87	22	192.168.10.90	44800	ESTABLISHED	431999

Table 3.2: Iptables connection table example (Src and Dst are acronyms for Source and Destination)

With these features, a packet filtering firewall provides an inexpensive type of network filtering. A single device can filter traffic for an entire network [8]. Packet filtering firewalls are not ideal for every network, so there are other types of firewalls.

Application-Level Gateway

The application-level gateway does not only filter packets by IP address or port but also by the application the packets originate from [6]. The application-level gateway is a server that sits between the application server and the client. The gateway acts as a server to the client and as a client to the application server. Since the application-level gateway serves as the communication partner for the client and the server, an address translation must take place in the gateway. This allows communication to take place between the client and the server. Each data stream is examined to decide whether the data should be forwarded, dropped, or marked for further examination [6]. It analyzes the data up to the application layer of the ISO/OSI model. By checking the data content, we can increase network security. Checking the data content makes it possible, for example, to allow access to a particular website but restrict individual pages [8]. The additional data stream analysis increases the processing time and thus affects the network's performance.

Next-Generation Firewall (NGFW)

NGFWs are a further development of the packet filter firewall. In addition to the normal packet filter functions, they offer further possibilities for packet processing. According to Güttich, NGFWs are firewalls that, in addition to the classic firewall functions, provide data analysis at the application level [9]. These data analysis functions include an Intrusion Prevention System (IPS). Furthermore, NGFWs include antivirus and anti-spam functions. An antivirus vendor usually provides the antivirus software, and the NGFW integrates them into the firewall [9]. NGFWs provide the ability to enable protection against Denial of Service (DoS) attacks, cross-site scripting, and Structured Query Language (SQL) injections. As Güttich states, powerful NGFWs can analyze the Secure Sockets Layer (SSL) and Secure Shell (SSH) connections [9]. They can further apply Uniform Resource Locator (URL) filters and scan downloaded files during or before delivery to the client. The benefit of an NGFW is the ability to track traffic from the data link layer up to the application layer. The various additional functions enable a wide range of usage. The additional functions increase the price of the NGFW compared to other firewalls. In addition, the performance of the hardware should be higher to enable all the functionality.

Circuit-Level Gateway

The circuit-level gateway enables session-level control of network traffic. Like the application-level gateway, the circuit-level gateway is located between the internal network and the outside world [10]. All connections from the internal network to the outside world go through the circuit-level gateway. The gateway takes over the address translation as with the application-level gateway. By monitoring TCP handshakes and other network protocol session initiation messages, the gateway decides whether the connection is trusted or not [8]. The circuit-level gateway does not check the packets themselves. This makes the circuit level gateway easy to set up and manage. This gateway type is often used in conjunction with other security systems [8]. For example, in combination with application-level gateways.

3.2 Linux Kernel Space Networking

In Linux distributions, the networking subsystem is not an essential component of the operating system kernel. Nevertheless, it is rather unusual to find a Linux system without a networking subsystem [11]. The kernel module implements the protocols up to the transport layer. The application layer protocols are often implemented in user space [11]. In this section, we describe the kernel modules necessary for our behavioral modeling of firewalls.

iptables

Part of the Linux kernel is the *iptables* module. *Iptables* is the most commonly used firewall software in Linux [12]. It is used to setup and maintain IP packet filter rules

in the Linux kernel [13]. To work with IP packet filter rules *iptables* uses the packet filter hooks in the Linux kernel. „These kernel hooks are known as the netfilter framework“ [12]. *Iptables* is, therefore, only a configuration interface for netfilter. Thereby *iptables* control three different chains. These are the *Input* chain, *Forward* chain, and *Output* chain, which *iptables* can configure. The *Input* chain controls incoming traffic. The *Forward* chain controls traffic that the device has to forward. This chain is essential for a firewall at the edge of a network. The *Output* chain controls the traffic that the device sends into the network.

The network device compares each packet of incoming and outgoing network traffic with one of the chains. If there is no matching rule for a packet, the default behavior of the firewall takes effect. With the help of *iptables*, we can define the default behavior. For example, a packet that does not match any rules of the chains can be dropped or accepted.

In general, *iptables* has three possible reaction types for a rule. A connection can be allowed by *Accept*. With *Drop*, the firewall drops packets without notifying the sender. With *Reject*, the firewall notifies the system, which tries to establish a connection that the firewall denies [13].

Netfilter

Netfilter provides the firewall functionality in Linux [12]. As a framework, *netfilter* provides various hooks with which modules like iptables can register. The hooks trigger at different points in the packet processing stack. Every incoming and outgoing packet triggers these hooks. The hooks a packet triggers depend on whether it is incoming or outgoing, what destination it has, and whether the packet was previously dropped or rejected [12]. *Netfilter* offers the following hooks:

- *NF_IP_PRE_ROUTING*: Before any routing decisions are made
- *NF_IP_LOCAL_IN*: The incoming packets destination is the local system.
- *NF_IP_FORWARD*: The incoming packet is destined for forwarding.
- *NF_IP_LOCAL_OUT*: The packet is destined for the outgoing chain.
- *NF_IP_POST_ROUTING*: This hook is triggered just before the packet is put out on the wire [12].

Modules that register with the hooks must specify a priority. The priority specifies the order in which the *netfilter* calls the modules when the corresponding hook triggers [12]. After *netfilter* calls the modules, they process the information and return to *netfilter* with instructions on what to do with the packet [12].

conntrack

Since iptables only serve as a user interface, netfilter creates and manages the actual stateful rules. The netfilter framework enables connection tracking through the *conntrack* module. *Conntrack* examines network packets to determine which

packets form a connection. In this process, *conntrack* only takes on an observer role. *Conntrack* observes connection-oriented protocols and connection-less protocols. It does not modify or drop the packets. From the observations, *conntrack* creates a connection table that contains all current connections. Other kernel modules can access these connection table entries and make decisions based on that [14].

3.3 Linux User Space Networking

Instead of using the Linux kernel for network communication, the possibility exists to communicate directly with the Network Interface Card (NIC). For this purpose, the networking is done through the user space. The *user space networking* aims to achieve high I/O performance and high packet processing rates. *User space networking* achieves higher I/O performance by communicating directly with the hardware instead of using the Linux networking stack [15]. Our thesis uses the Data Plane Development Kit (DPDK) framework for fast packet processing and the FD.io Vector Packet Processing (VPP) network stack. We first describe DPDK and then FD.io VPP to better understand their background and function.

Data Plane Development Kit (DPDK)

DPDK is a *user-space packet I/O* framework and provides the ability to develop efficient network applications. It allows the direct exchange of Ethernet frames with the user space without involving the Linux kernel [16]. In Figure 3.1, we compare the classic network communication with the network communication of a DPDK application. We observe that standard applications communicate with the NIC via the kernel. DPDK applications communicate with the NIC via Poll-Mode-Driver (PMD). The PMD enables the direct sending and receiving of data packets. The PMD polls the NIC cyclically for new data packets [16]. The cyclic polling prevents interrupts from the NIC when new data packets arrive. These interrupts usually cause additional overhead.

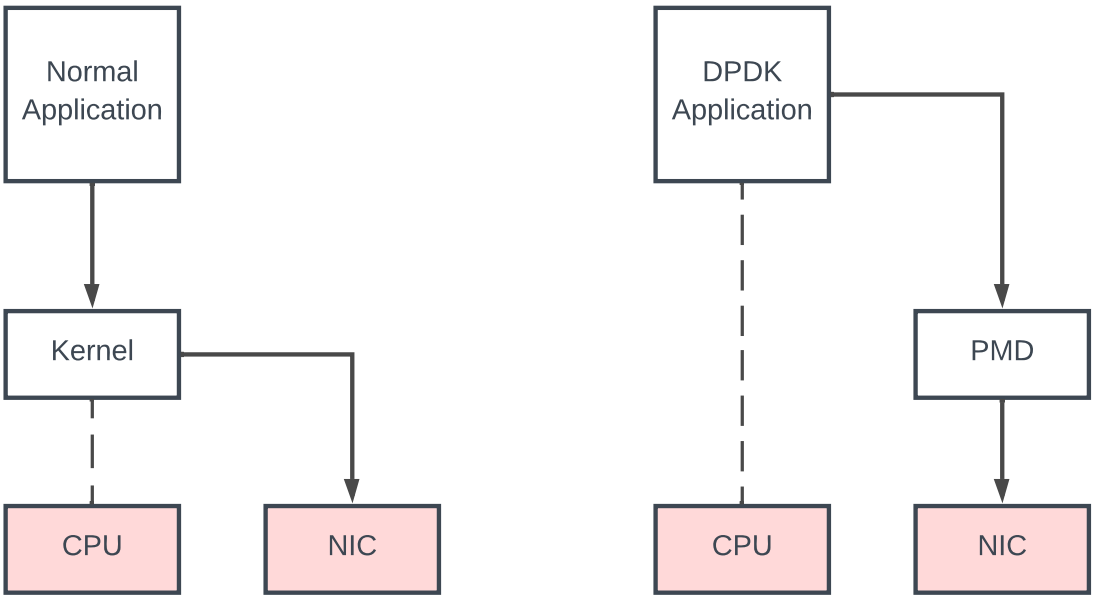


Figure 3.1: Comparison between normal network applications and DPDK applications. Based on [16]

FD.io VPP

FD.io VPP further referred to as VPP, is a high-performance network stack [17]. VPP offers switching and routing functionality. We treat VPP in this thesis as part of our DUTs. VPP uses DPDK as driver for the NIC communication to process the packets with high performance [18]. VPP can run on standard central processing units (CPUs). The way of packet processing without strict performance requirements is scalar packet processing [19]. With scalar packet processing, a „interrupt handling function takes a single packet from a network interface and processes it through a set of functions“ [19]. This way of processing is simple but inefficient. With the overuse of the systems memory thrashing occurs. Each packet suffers a similar number of instruction-cache[1] misses [19]. VPP instead uses vector processing. A vector processing network stack processes up to 256 packets simultaneously. In this processing, an „interrupt handling function takes the vector of packets from a Network Interface and processes the vector through a set of functions“ [19]. This spreads the cost of instruction-cache loads over multiple packets and leads to faster processing.

[1] „The instruction cache is used to keep recently executed instructions closer to the processor if those particular instructions are executed again“ [20]

3.4 Temperature Influence on Computer Performance

Every computer hardware produces heat through its use. Too much heat can cause a slowdown of the computer system [21]. „If the CPU temperature is too high, for example, a mechanism will trigger that reduces performance in order to avoid damaging the processor“ [21]. This safety mechanism causes the frequency to change dynamically. As a result, there are performance losses. One reason for this is that electrical conductors have a specific internal resistance. The conductors' resistance is temperature-dependent [22]. The increased resistance disturbs the electron flow. The reduced flow of electrons leads to a decrease in the performance of the device.

3.5 Letter-Value Plot

Letter-value plots use recursively defined boxes to visualize the different dataset partitions. In Figure 3.2, we present an example letter-value plot. The black line next to zero represents the median. Each letter value has a specific depth that we calculate recursively, starting with the median:

$$d_1 = \frac{(1+n)}{2} \tag{3.1}$$

n is the number of data points. All successive letter values are defined by:

$$d_i = \frac{(1+\lfloor d_{i-1} \rfloor)}{2} \tag{3.2}$$

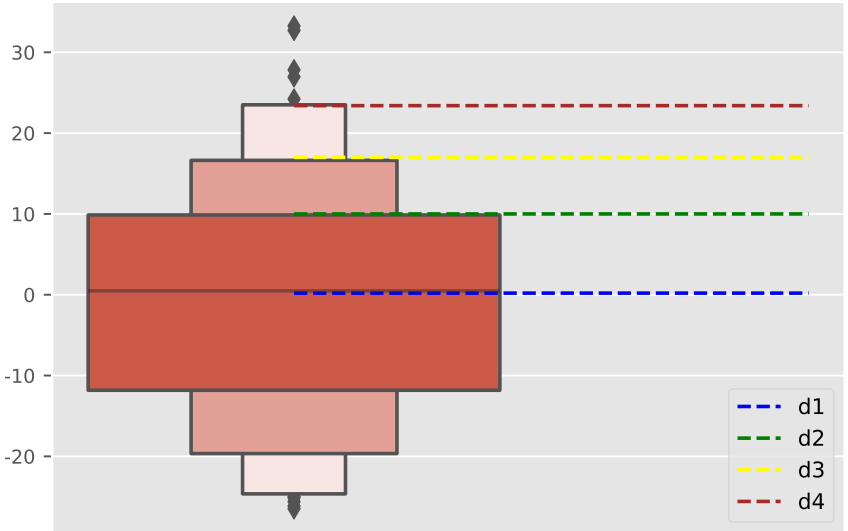


Figure 3.2: Example Letter-value plot

„Because each depth is roughly half the previous depth, the letter values approximate the quantiles corresponding to tail areas of 2^{-i} “ [23]. The stopping criterion for calculating the boxes is when the 95 % confidence interval overlaps with the subsequent letter value. Boxes are only shown for letter values whose 95 % confidence interval excludes neighboring letter values [23]. All points outside the boxes are displayed individually as outliers.

3.6 Runge's Phenomenon

Runge's phenomenon describes a property of polynomial interpolation. A higher degree of the interpolation polynomial leads to worse interpolation quality. In Figure 3.3, we demonstrate that an unfavorable choice of interpolation points and a too high polynomial degree can lead to a worse interpolation result. The actual distribution function is therefore not correctly reproduced. The interpolated function starts to oscillate [24]. Polynomials of higher degree, therefore, usually have a noticeable interpolation error over the entire interval. Schmid [24] mentions in his book, to prevent the phenomenon, it is better to use piece-wise composite polynomials of degree three.

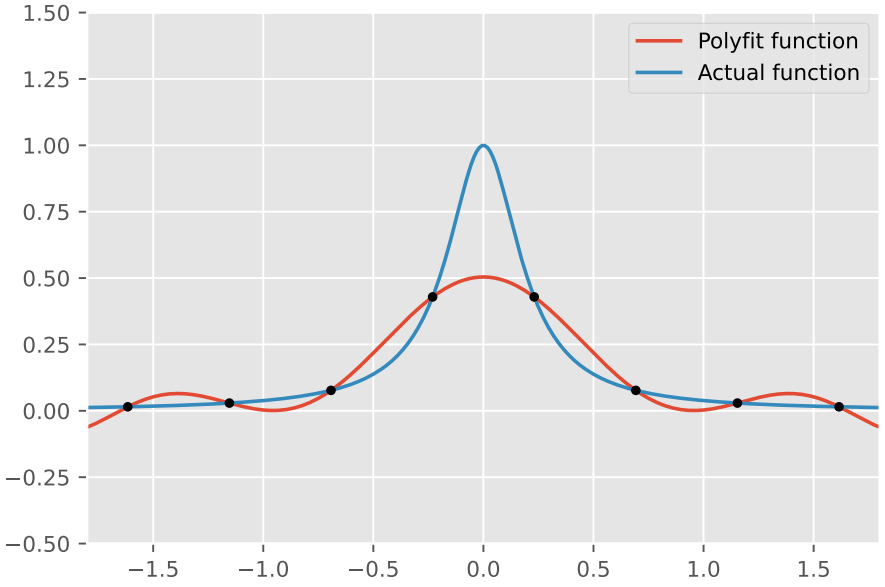


Figure 3.3: Example plot for Runge's phenomenon

3.7 Python SciPy Curve Fit

In our thesis, we model the behavior of the DUT by using a function. Python offers with the SciPy library the possibility to fit a given function to data. The function for this is called *curve fit*.

Curve fitting is an optimization problem. The goal is to find parameters for a defined function that fits the data best [25]. We define the function ourselves. Depending on the data we measure, we can define different functions to model the behavior. The *curve fit* function uses non-linear least squares to fit the function to the data. The idea is to choose the parameters so that the distance between the data and the curve is minimal.

3.8 Objective Modular Network Testbed in C++(OMNeT++)

OMNeT++ is a simulation library and framework for creating network simulators [26]. *OMNeT++* is modular and can be used for different applications. We can map the following examples with *OMNeT++* [27]:

- telecommunication networks modeling
- protocol modeling
- model multiprocessors and distributed hardware systems
- validate hardware architectures
- evaluate performance aspects of complex software systems

In the context of our thesis, it makes sense to use *OMNeT++* for traffic modeling of networks and evaluating performance aspects.

3.9 Network Security Architecture

In the following, we describe common network architectures that define secure network design for industrial automation. We show with the architectures how essential network security devices are. In addition, we show the role they play concerning the performance of a network. First, we present the Industrial Control System (ICS) network architecture. Second, we describe one of the most common industry network patterns called zones and conduits. We then describe how to set up a defense-in-depth network.

3.9.1 Industrial Control System (ICS) Network Architecture

An ICS consists of several control systems that operate together. Figure 3.4 shows a simplified variant of an ICS network. The ICS network has a hierarchical structure.

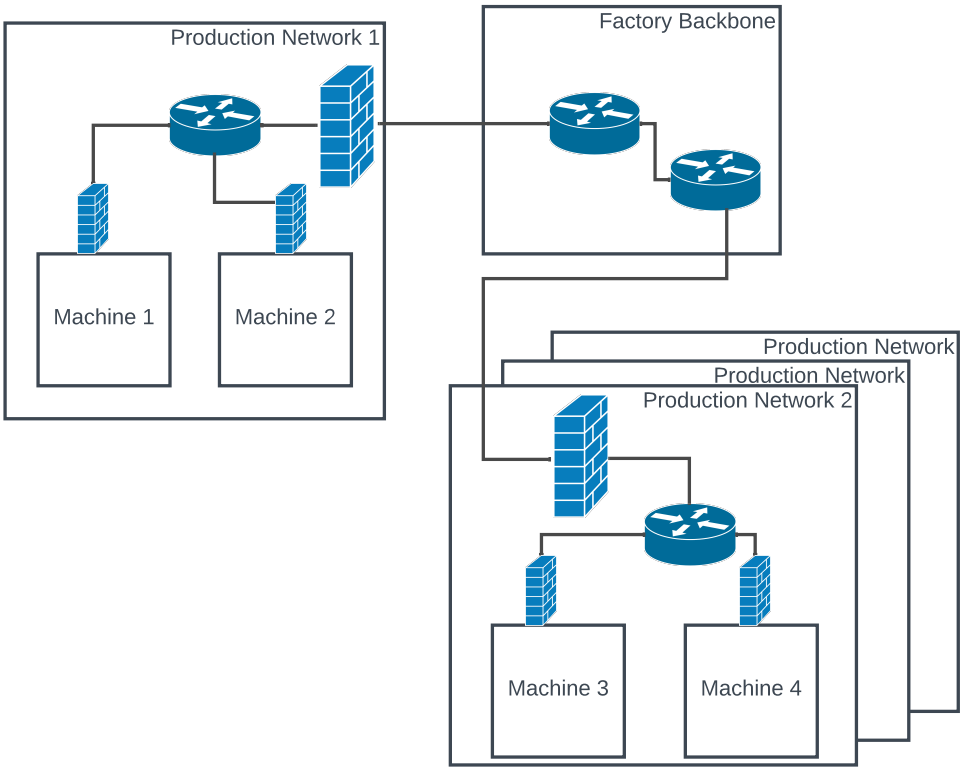


Figure 3.4: Example ICS network with firewalls for access control based on [28]

In Figure 3.4, the factory backbone network connects an arbitrary number of separate production networks with each other. Each production network can be further divided into individual machine networks. There is a firewall for access control at each transition into a subnetwork. The companies often use software firewalls to keep the networks as cheap as possible. The production networks use TSN for real-time traffic. TSN contains different types of time-critical traffic. Following IEC/IEEE 60802 [29] and the work of Wüsteney et al. [2], we distinguish between three types of time-critical traffic:

- Isochronous traffic
- Cyclic traffic
- Acyclic traffic

Isochronous traffic consists of packets that repeat at fixed time intervals. The cycle times are less than 2 ms [2]. Isochronous traffic is often used within the machine network. With cyclic traffic, the cycle time is 2 ms to 20 ms [2]. Acyclic traffic does not occur at fixed repeating intervals and therefore does not have a cycle time. Alarm or event notifications count as acyclic traffic. Packet loss is not tolerated to ensure that alarms and events reach their destination. Cyclic and acyclic traffic occur across multiple networks.

3.9.2 Zones and Conduits

A fundamental concept of network security in industrial networks is the concept of *zones and conduits* [30]. Part of this pattern is connecting multiple devices or systems with the same security level in zones. We define physical zones based on their physical location. For logical zones, we group the systems according to their functionality or characteristics. The network administrator can realize these zones, for example, by using VLAN. The different zones communicate with each other through conduits. According to Knapp et al. [30], security conduits are a particular type of zone that groups communications. The conduits manage the information flow between zones. Conduits contain security measures to control the data flow and access. The *zones and conduits* pattern can limit communication so that the zones become more secure. If a threat exploits a vulnerability within a zone, this pattern can limit the negative consequences [30]. Knapp et al. [30] state that networks are, in reality, divided into only a few zones. This means that the level of security decreases. The more fine-grained the zones are, the higher the level of security.

For access control in conduits, we can use a firewall. Since a conduit is the only communication interface of a zone to the outside world, the firewall's performance is essential. It can form a bottleneck in such a structure.

3.9.3 Defense-in-depth

With a series of defensive mechanisms, the defense-in-depth tries to protect a computer network [31]. When a security mechanism is compromised, another is ready to stop the attacker. Since there are countless possibilities of attacks, no single method stops all attacks. Defense-in-depth allows reducing the risk of a successful attack.

In a network, for example, a firewall, an IPS, and additional antivirus programs can act as a *defense-in-depth* architecture. If the attacker passes the firewall, the IPS can detect and prevent the attack. If the attacker also gets past the IPS, the antivirus program is still on the computers. This example is not absolute protection, but the multiple barriers make a successful attack more difficult.

In *defense-in-depth* architectures, we can also use firewalls. Since the security mechanisms are built one after the other, the performance of the firewall alone is not decisive for the network performance [31]. However, even in this example, the firewall is a security layer where it is essential to know how it affects the network performance. Since the packets traverse each layer, they affect the performance.

4 Related Work

In this chapter, we explain the state-of-the-art research. First, we show and discuss related work that examines the behavior of firewalls. Second, we demonstrate the measurement environments of other research measuring latency, jitter, and packet loss. Finally, we provide an overview of performance studies for other network security devices.

4.1 Firewall behavior

Firewalls are a fundamental component of network security in industrial networks. Industrial networks and real-time traffic have certain behavioral requirements for the devices that operate in the network. In this section, we describe what related work has studied the behavior of firewalls and demonstrate why it is important that we examine the firewall behavior in terms of latency, jitter, and packet loss.

RFC3511 „Benchmarking Methodology for Firewall Performance“ defines general benchmarking methods for firewalls [32]. The tests in the RFC3511 can help to characterize the throughput performance of firewalls. Balarajah et al. [33] develop benchmarking methods to determine the performance of firewalls and make RFC3511 obsolete. The authors show that especially NGFW have many security features that affect the performance of the firewall differently. The benchmarking methods are mainly aimed at determining the firewall's performance in terms of throughput. The jitter or packet loss behavior is not considered in detail.

Zvabva et al. [34] consider the performance issues of industrial firewalls. Due to the network segmentation of the industrial networks, firewalls are used at every network border and are therefore a fundamental part of the network communication of industrial networks. In their evaluation, Zvabva et al. [34] address the latency, jitter, and packet loss generated by open source Linux firewalls in Modbus TCP/IP. They find that in Modbus TCP/IP, the maximum latency and jitter increase proportionally to the number of general firewall rules. In their measurements, they observe no packet loss. From their results, they conclude that if the IEC 62443 security standard is strictly followed, some time-sensitive traffic is not possible. We note with the work of Zvabva et al. [34] that there is already research in the industrial environment on the behavior of firewalls but this work is only related to Modbus TCP/IP. The findings from their work show that the firewall has a great influence on the network performance and that some time-sensitive communication through firewalls is not possible. In contrast to our work, they did not attempt to model the behavior of firewalls.

Cheminod et al. [35] present an approach to obtain information about the effects of industrial firewalls in networks. The paper of Cheminod et al. [35] also focuses on the performance causes of industrial firewalls on Modbus TCP/IP. In contrast to

Zvabva et al. [34], Cheminod et al. [35] investigate the performance effects on latency and bandwidth. They do not give an estimate of the real-time traffic capability of industrial firewalls.

In the paper „Performance Evaluation and Modeling of an Industrial Application-Layer Firewall“ Cheminod et al. [36] investigate how the performance of a firewall with Modbus TCP/IP traffic can be modeled. Cheminod et al. [36] focus on packet latency as a significant performance index. They show in their work that latency performance modeling of COTS devices for Modbus TCP/IP is possible and provides benefits for system designers. For our modeling, we consider latency, jitter, and packet loss, because latency alone is not sufficient to provide sufficient information about the suitability of firewalls in industrial networks with real-time traffic.

A fundamental security pattern for industrial networks is the zones and conduits pattern [2]. The network under consideration is divided into zones, i.e., VLANs and subnets. These zones are connected via conduits. Conduits are specific gateways equipped with security measures to control access, i.e., firewalls and switches with Access Control Lists (ACLs). A key mechanism for improving network security are packet filters. Based on the digitalization trend in industrial networks, delay and jitter became an increasing problem in such networks, especially with time sensitivity. TSN is used for realizing advanced real-time apps in industrial networks. In "Impact of Packet Filtering on Time-Sensitive Networking Traffic" [2] published by Wüsteney et al. [2], the authors analyze problems that arise with the segmentation of TSN Networks. ICS networks require segmentation provided by zones and conduits. The network segmentation leads to a conflict between the security and performance of the network. At the time of our work, there is no firewall that we can use in a TSN network [1]. However, new control possibilities in the industry make this necessary. Some applications already offer control from the cloud, where the packets have to be sent across several subnets. Wüsteney et al. [2] measurements show that as the number of processing steps increases, the jitter increases non-linear. The increasing jitter leads to more complex forwarding models for the jitter. Furthermore, the measurement results show that the faster CPU of a test firewall leads to more predictable delays. Wüsteney et al. [2] conclude that packet filtering and forwarding on general-purpose CPUs is not deterministic and difficult to predict [2]. The results suggest that ACL based packet filters fit very well to TSN traffic because the delay range and the delay behave constantly. The disadvantage of devices with ACL rules is that they only support relatively few ACL rules. The work of Wüsteney et al. [2] shows that the research is addressing the impact of firewalls on TSN traffic. In contrast to our work, the authors did not model the latency, jitter, or packet loss behavior. The authors show that the compatibility of firewalls with real-time TSN is mainly influenced by the delay and jitter of the firewall. Wüsteney et al. [2] show that it is important for the real-time capability of firewalls to obtain information on latency, jitter, and packet loss.

The related work shows that the topic of firewall behavior prediction in industrial networks is important. The estimation of whether a firewall can be used in an industrial network with real-time traffic or not depends on its latency, jitter, and packet loss. Previous related work focuses on throughput, latency, firewall behavior in Modbus TCP/IP, and does not model the behavior to an extent necessary for the

use in industrial networks. Our model provides the firewall behavior information to quantify the industrial network and real-time capabilities of firewalls.

4.2 Measurement setups

In this section, we examine the measurement setup, measurement accuracy, and modeling for latency, jitter, and packet loss in related work.

Programming languages such as P4, which enable the control of packet forwarding planes, lead to new applications in the data plane. The new applications make it difficult to define performance expectations. Scholz et al. [37] create a framework to analyze and model the performance of P4 program components.

The modeling framework of Scholz et al. [37] records performance metrics through the load generator and the DUT. The load generator measures throughput, packet rate, and latency. The DUT, in turn, records CPU cycle usage, cache misses, and resource consumption [37]. For the measurements, the authors specify network traffic and parameters for the DUT. Initially, the authors measure the DUT at the maximum packet rate, then at 10 %, 50 %, and 70 % of the maximum packet rate. Scholz et al. [37] use *MoonGen* for the generation of the traffics. In our work, we also use *MoonGen* as a packet generator to perform the measurements. With *MoonGen*, we can write individual packet generator scripts to generate different network scenarios. To create a model from the measured data, Scholz et al. [37] use the curve fit function of the Python SciPy module. We also use the curve fit function in our modeling because we can fit functions to the measured data and use the functions for predictions.

Scholz et al. [37] state that their framework is limited by the target dependency. Especially for software targets, the model derived for one platform may no longer be valid for another platform. Scholz et al. [37] state that the accuracy of their model is determined by the number of data points used for curve fitting. Accordingly, we examine how the number of measurement points affects our model.

From Scholz et al. [37], we deduce that modeling with the help of curve fit is common and that we can include such modeling in our work. In addition, the authors mention that the same software delivers different results on different platforms. The influence of different platforms is also a factor we have to consider in our modeling.

Harkous et al. [38] describe the influence of P4 constructs on packet processing latency. From the influencing factors, the authors create a method to predict the packet latency on P4-based network functions [38]. Since on P4 targets, each stage of the P4 pipeline can have different features and building blocks, a complete examination of all combinations is impossible [38]. Harkous et al. [38] start by measuring the simplest P4 case and gradually increase the pipeline complexity. With the measured results, the authors create a packet latency prediction method for different P4 programs. The testbed setup on which the measurements are performed consists of two servers that are connected. On one of the two servers *MoonGen* is installed as a packet generator. The other server contains the Smart NIC with the various P4 programs to be tested. A Smart NIC is a programmable network card to speed up network processing. Harkous et al. [38] define in their work the

generated packet size with 1000 Bytes and the data rate of 10 Gbps. *MoonGen* is used for packet generation, measuring latencies, and reporting results. Exactly four P4 programs are tested to establish the prediction method. Based on these results, the authors explain that the average packet latency is composed of the time for parsing and modifying headers and the number of tables in the P4 pipeline [38]. To validate their results, they used their method to predict packet latencies for two realistic network functions. They measure the same network functions in reality and compare the measurement results with their prediction. The deviation resulting from this determines the accuracy of the prediction method.

The work of Harkous et al. [38] shows that other scientific groups are addressing the issue of latency prediction. Also, Harkous et al. [38] use *MoonGen* to generate their test traffic. However, only certain P4 functions are considered and not network security devices as a whole. In contrast to the work of Harkous et al. [38], we consider software firewalls as a whole and model their behavior.

The measurement accuracy is important to generate reliable measurement results. Therefore, Wüsteney et al. [2] state that their synchronization accuracy between two time stamp units is 30 ns based on the IEEE 802.1AS. Time stamps are necessary to determine the latency of the DUT. The time stamps determine the time before the DUT processes the packet and after. The measurement accuracy of Wüsteney et al. [2] is sufficient because the delay and jitter in each measurement are tenfold higher than the precision. Furthermore, TSN is based on the same IEEE 802.1AS precision. For our time stamping, we use only one time stamping switch that sets the packet time stamps in its ingress or egress. The time stamping switch sets the time stamps in the packet payload. Our time stamping method is more precise, as it eliminates the time synchronization error.

In conclusion, we observe that MoonGen is widely used to generate network traffic in measurements. MoonGen offers many advantages for dynamic traffic generation and analysis that we can use in our thesis. For our modeling, we use as well as other related work Python SciPy curve fit.

4.3 General network security device behavior measurements

In this section, we examine related work that generally investigates network security devices on latency, jitter, and packet loss. From the approach of other investigations, we derive important criteria and aspects for our research.

Pudelko et al. [39] published a "Performance Analysis of VPN Gateways". In their research, they show the dependencies on CPU load, packet rate, and the number of flows of different Virtual Private Network (VPN) Gateways. In their paper, they test IPsec, OpenVPN, and WireGuard. They analyze which VPN implementation is fast enough to be installed on a 40 GBit/s link with components-off-the-shelf (COTS) hardware [39]. The authors use a separate server as a load generator on which specially written *MoonGen* scripts run. For our work, we observe that other network security device performance research uses *MoonGen* to generate network traffic. Initially, Pudelko et al. [39] create a baseline measurement to determine if

a bottleneck is caused by an application or the Linux network stack. The ideal case would be if every packet is forwarded. In our thesis, we create a baseline measurement as well to explore the general behavior of the firewall without firewall rules.

The benchmark results of Pudelko et al. [39] show that none of the three open-source software VPN implementations is fast enough for the 40 GBit/s test network. Part of the overhead comes from the Linux network stack on which the three variants are based. The authors' DPDK-based VPN implementation shows that significant performance gains can be achieved through kernel bypassing. They show that DPDK changes the behavior of the VPN. In our work, we investigate DPDK-based firewalls. We, therefore, expect a change of the DPDK firewall behavior compared to iptables firewalls.

Hasan et al. [40] propose a latency-aware trust system placement. A trust system is a specialized security device that includes firewalling and intrusion detection. The Supervisory control and data acquisition (SCADA) communication network is part of a smart grid. Trust systems are used especially in the SCADA networks to protect the network against cyber-attacks. The placement scheme of Hasan et al. [40] tries to place all trust systems in the SCADA network to keep the latency below a certain threshold. For our work, Hasan et al. [40] findings showed that latency determination and optimization are necessary not only in industrial networks but also in smart grids. In contrast to our work, they only consider latency.

The related work shows that the investigation and modeling of latency are not only necessary for firewalls but also for other network security devices to meet the requirements in their respective networks. We also show that several research papers on the performance of network devices use *MoonGen*. Due to the common use of *MoonGen*, we also use it for our network traffic generation.

5 Design

In this chapter, we describe the system model of our latency and packet loss test environment. We describe the essential factors for our system model. First, we define the network environment for our measurement setup. Second, we determine the essential network parameters for our latency and packet loss model. Third, we present our measurement setup to profile the firewalls and test our model. Next, we describe the two types of modeling that we consider. Afterward, we define how to construct the model and how we profile the DUTs. Finally, we discuss our derived simulation model.

5.1 System Model

Our system model for latency and packet loss modeling describes a general network setup. Based on this, we formulate the test cases for the DUTs. Realistic network traffic is part of this system model for latency and packet loss modeling. To simulate a realistic network behavior, we first need information about what realistic network traffic can look like. The Simple Internet Mix (IMIX) describes a general Internet traffic pattern based on the average packet sizes that occur. Network device vendors compare their devices using Simple IMIX. Accordingly, the Simple IMIX is a reasonable basis for our packet generation.

Packet size (incl. IP header)	**Number of packets**	**Distribution in packets**	**Bytes**	**Distribution in bytes**
40	7	58.33%	280	7%
576	4	33.33%	2304	56%
1500	1	8.33%	1500	37%

Table 5.1: Simple IMIX packet size distribution

Table 5.1 represents the distribution of various packet sizes in a network, according to Simple IMIX [41]. The distribution shows that there are many small packets, a few medium-sized packets, and very few large packets in the network traffic. The Simple IMIX gives us an indication of the size distribution we can expect. Based on this, we can make our system model realistic.

To make the network traffic more realistic, we consider changing and constant data rate of the packets. In other words, a network offers a maximum data rate, but realistic network traffic does not use a constant data rate. We assume that the data rate can change during a measurement run to include this behavior in our system model.

Since we are looking at software firewalls and want to model their latency and packet loss, we also vary the firewall rule configurations. Both stateful and stateless rule configurations are allowed on the firewall. For iptables configurations, note that stateful rules include a connection timeout. This connection timeout is relevant for the latency consideration as it produces additional overhead. The timeout depends on the first received packet of that flow. For example, if we have already established a TCP connection, the default timeout is five days. If the connection is currently in the middle of the TCP 3-way handshake, iptables chooses 120 seconds as the timeout limit [7]. Moreover, the limit can be modified arbitrarily by the administrator.

Another aspect of creating real network traffic is the choice of the packet transport protocol. We allow TCP and UDP in our system model so that our resulting model is not dependent on the chosen transport protocol.

In a network environment that is as realistic as possible, the firewalls face dynamic reconfiguration. Accordingly, in our system model, we also consider CPU load that does not arise from forwarding packets. We can create the additional CPU load through SSH connections, access via the web interface, or the monitoring of the devices.

5.2 Dependency Tree

We created a dependency tree to determine a possible latency dependency between several DUT and network parameters. Network parameters are measurable values that we examine for our modeling. We show the most important parameters for the behavior influence of DUTs in Figure 5.1. We use the dependency tree to determine the effects of specific parameters on the DUT latency. Various network scenarios and external influences play a role in our dependency considerations.

Our goal is to develop a model that considers all necessary parameters. That results in a comprehensive picture of the corresponding DUT. In Figure 5.1, we present the parameters in tree form. The leaves of our dependency tree represent all the influencing parameters we are investigating. All nodes above the leaves group the parameters into appropriate groups. Each group represents a part of the prediction function. The root in Figure 5.1 corresponds to our latency function for the DUT.

We show that internal and external parameters determine the device performance on network traffic. Internal parameters are influenced by the DUT, such as the number of firewall rules. External parameters determine the network traffic, such as the data rate. We divide the dependency root into two parameter groups. On the bottom, we have the temperature factor, and on the top, we have the device factor. The temperature factor describes the ambient and device temperature. We know from dealing with other electronic hardware that they need some cooling to ensure functionality. This leads us to the assumption that the temperature influences the latency. In our investigations, we measure whether temperature changes influence the devices. We consider a possible positive influence due to unfavorable temperatures and a negative impact due to high temperatures. Industrial firewalls, such as the EAGLE40, are specified for a particular temperature operating window by the manufacturer. The maximum permissible temperature indicates that

the device is still functioning up to this point. The EAGLE40, precisely, is specified with an operating window up to 105 degrees Celsius [42]. In our measurements in Section 6.4, we investigated the effect of temperature on latency and packet loss. Our measurements show that within the operating window, we can guarantee the functionality of the device. We did not measure a relevant latency effect, so we marked the temperature factor in Figure 5.1 red. If the temperature exceeds the permissible limit, we cannot predict the behavior of the DUT since the CPU frequency abruptly throttles. More about the influence of temperature on latency in Section 6.4.

Other external influencing parameters, such as humidity, cannot be investigated within the scope of this work. On the one hand, this is because the devices could suffer damage by excessive humidity, which would falsify future measurements. On the other hand, we cannot set the humidity in our measurement setup with sufficient accuracy.

The Device Factor is a combination of multiple different factors. The Device Factor indirectly determines the DUTs hardware performance. As the performance of the device improves, it is more likely that the DUT produces lower latency. We classify our device factor by several parameters:

- Base load
- Packets that need to be processed
- Configured firewall rules
- Device management

We specify the base load as a constant in each measurement. This is the minimum time a packet needs to traverse through the DUT, shown as a red arrow in Figure 5.2. For the determination of the base load is, no rules configured, no other packets in the firewall, and no additional CPU load on the firewall. Since the base load is always present, its latency is present in every measurement. Furthermore, we are interested in the variable factors and in being able to predict them as precisely as possible. To do this, we need to know all these factors.

The variable factors include the packet factor. We determine the packet factor by the distribution of the packet size, the number of packets per second, and the selected transport protocol. To be able to represent network traffic as realistic as possible, we choose a packet distribution function that can represent the Simple IMIX. In addition, we expect that the chosen distribution function should be able to map other packet size ranges besides the Simple IMIX distribution. One of the most influential factors is the *number of packets per second* parameter that combines the data rate and packet size. We show later in Equation (6.1) how we calculate the number of packets per second. Due to network traffic, this factor significantly determines the load on the DUT.

The last part of our packet factor is the selected transport protocol. As mentioned in Section 5.1, both TCP and UDP traffic is allowed. It is not necessary to distinguish between transport protocols because the firewall considers the header of the packets in its processing. A TCP header is slightly larger than a UDP header. Accordingly,

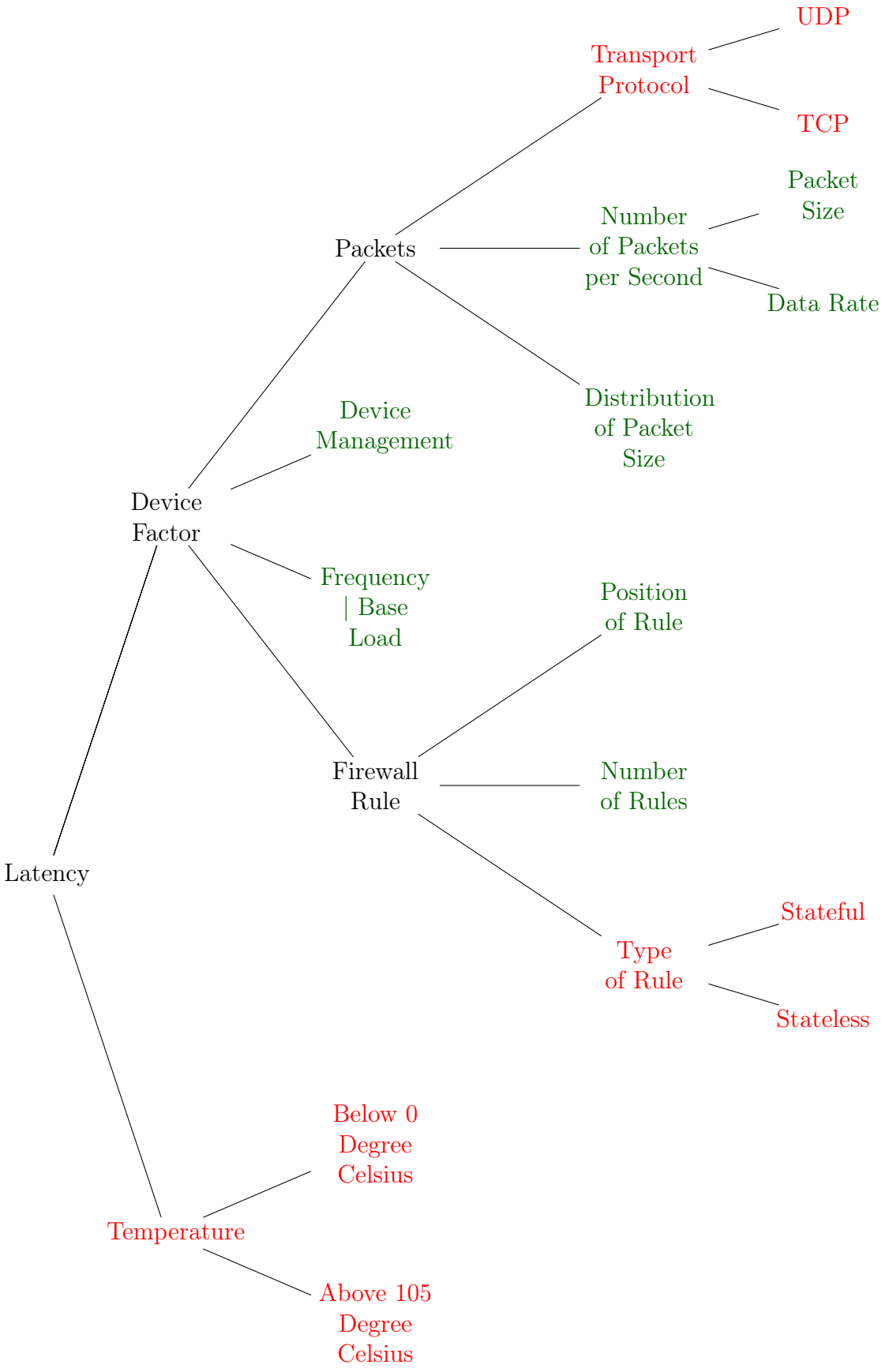


Figure 5.1: Dependency tree for determining the latency composition

the two protocols do not differ in latency. We examine this in more detail in Section 6.1.4. As a result, we omit the transport protocol factor in our model and mark it red in the dependency tree.

In addition to the factors mentioned so far, we consider the influence of firewall rules on latency. Their impact depends on their number, type, and position. On average, the more firewall rules we configure before our matching rule, the longer it takes for a packet to match with the corresponding rule. The worst-case scenario for this factor is a packet that has to be compared with each rule to find out that the default rule applies. The position of the matching rule plays a decisive role. If the matching rule is always the first rule in the filter table, it does not matter how many firewall rules we configure afterward. The remaining rules are no longer taken into account by the firewall. We consider in our further modeling the average matching position of the firewall rules because different packets match at different firewall rule positions. As an example, we assume 100 configured rules. The incoming traffic matches exclusively to the first and last rule position. If the assumed traffic distributes evenly, the average matching position is 50. In other words, each packet is compared with an average of 50 rules until a match occurs.

The firewall rules factor also includes whether it is a stateful or stateless rule. It depends on the selected firewall whether differentiation of the rule type causes a latency change. The implementations and thus also the function of the rule types differ depending on the software firewall type. We describe the differences in more detail in Section 6.2.4.

To collect the data, we define in the next section what our measurement setup looks like.

5.3 Measurement Setup

In this section, we explain the measurement setup for the latency and packet loss measurement in different firewall configurations and network scenarios. We use this measurement setup to profile the behavior of various software firewalls and verify and create our model. We test different software firewalls, the analyzed firewalls are referred to as DUT in the respective measurement. To validate our model, we create random test scenarios consisting of different data rates and firewall configurations.

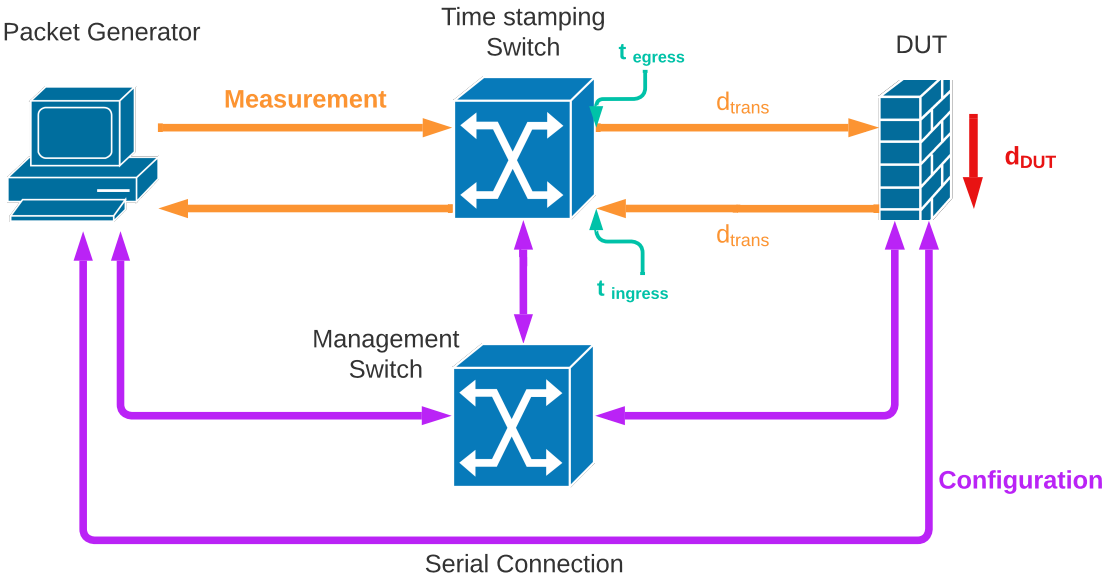


Figure 5.2: Measurement Setup (arrows show the flow direction)

The test bed consists of several interconnected devices. Figure 5.2 shows the setup schematically. The packet generator generates the test traffic and calculates the latency and packet loss. Within a measurement, the packet generator sends packets to the time stamping switch. We place the time stamping switch between the packet generator and the DUT, as we show, in Figure 5.2. In our setup, the time stamping switch inserts the time stamps in the payload of the generated packets. The first time stamp is set at the egress of the time stamping switch, called t_{egress} in Figure 5.2. In more detail, we explain the time stamping in Section 5.3.2. Our time stamping switch forwards the packets to the DUT. After processing the packets, our DUT sends these packets back to the time stamping switch. The second time stamp is set at the ingress of the time stamping switch, called $t_{ingress}$ in Figure 5.2. The time stamping switch sends the packet with both time stamps back to the packet generator. The packet generator evaluates the packet.

5.3.1 Device Under Test (DUT)

In our test setup, we examine several firewalls. Table 5.2 shows the different hardware and software combinations along with their names. We not only check the behavior of different hardware with the same software firewall but also look at the same hardware with two different software firewalls on it. The devices we tested include an EAGLE40 industrial firewall from the Hirschmann Automation and Control GmbH with a standard Ubuntu 20.04.3 LTS installed [43]. We examine both standard iptables (DUT 1) and VPP (DUT 3) on this firewall. In addition, we inspect an EAGLE30 (DUT 2) from the Hirschmann company. The unmodified EAGLE40 usually uses another operating system than our Ubuntu 20.04.3 LTS. We

use Ubuntu as the operating system on the EAGLE40 because we can test different software firewalls on the same device.

DUT Number	Hardware	Software
DUT 1	EAGLE40	iptables
DUT 2	EAGLE30	iptables
DUT 3	EAGLE40	FD.io VPP

Table 5.2: Overview of the DUTs in their hardware and software combination

5.3.2 Time Stamping

We use an RSPE35 industrial Ethernet switch from Hirschmann to insert the time stamps. Since the switch's default configuration does not offer time stamping functionality, Hirschmann provided us with a modified version that allows us to set time stamps. As shown in Figure 5.2, we locate the time stamping switch between the packet generator and the DUT. The RSPE35 inserts time stamps to the payload of the packets. The RSPE35 sets the first time stamp on its egress, referred to as t_{egress} in Figure 5.2. After the packet returns from the DUT, the switch sets the second time stamp at its ingress. In Figure 5.2, this happens at $t_{ingress}$. Setting the time stamp does not affect the measured latency because the time stamping switch sets the time stamps in hardware at its egress or ingress. Therefore, we do not influence the DUT, and thus we do not falsify the latency measurement.

The difference between t_{egress} and $t_{ingress}$ determines the measured latency $d_{latency}$. The latency $d_{latency}$ is the time it takes for a packet to travel from the RSPE35 egress t_{egress} to the DUT and back to the RSPE35 ingress, $t_{ingress}$. Figure 5.2 shows that we create t_{egress} at the egress of the time stamping switch and $t_{ingress}$ at the ingress of the time stamping switch.

The time stamps are 4 bytes in size and have an accuracy of nanoseconds. This allows us to represent a maximum of $2^{32}Bit = 4,294,967,296\ ns$. To be able to measure longer than 4 seconds, we use the time stamps only to represent the decimal places of a second. Accordingly, each time stamp is between 0 μs and 999,999.999 μs. If the time stamp overflow occurs during a packet transmission between the time stamping switch and DUT, the ingress time is less than the egress time that the RSPE35 has assigned to the packet. In this case, we adjust the calculation of our latency to avoid wrong results. Equation (5.1) shows how we proceed in these cases.

$$d_{latency} = t_{ingress} + 1,000,000\ \mu s - t_{egress} \tag{5.1}$$

The latency $d_{latency}$ contains the transmission time of the packet. We show the transmission time in microseconds in Figure 5.2 as d_{trans} (orange arrows) between the time stamping switch and the DUT. In Figure 5.2, we show with a red arrow the actual latency d_{DUT} of the DUT. We want to measure only the device latency for our modeling. To determine the device latency, we need to calculate the packet transmission time. We calculate the packet transmission time, d_{trans} as follows:

$$d_{trans} = \frac{(s_{packet} + 20\ Byte) \cdot 8}{r_{max}} \tag{5.2}$$

The frame size s_{packet} for each packet is given in bytes. In Figure 5.3, we show the composition of the layer 1 frame. The preamble, the start frame delimiter, and the layer 2 Ethernet frame together form the layer 1 frame. The interpacket gap is placed between the layer 1 frames before they are sent. Therefore, we add 20 bytes to the Ethernet frame size.

Figure 5.3: Structure of a layer 1 frame

r_{max} describes the link speed and is specified in MBit/s.

We display the packet transmission time, d_{trans}, in Figure 5.2 between the time stamping switch and the DUT. The packet transmission time occurs on the outbound and inbound paths. Equation (5.3) shows that we subtract the packet transmission latency d_{trans} twice from the measured latency $d_{latency}$ to receive the actual latency d_{DUT}. The latency shown as a red arrow in Figure 5.2 is thus specified by d_{DUT}.

$$d_{DUT} = d_{latency} - 2 \cdot d_{trans} \tag{5.3}$$

5.3.3 Packet Generator

The packet generator is responsible for generating packets to imitate realistic network behavior. We display the packet flow in Figure 5.2 as orange arrows. The generator also handles the evaluation of the time stamps and thus acts as the start and destination point of the packets. After reading the time stamps from the packets, we discard the packets.

We use a personal computer with an Intel Core i9-9900K CPU and 32 GB RAM as packet generator. The packet generator requires DPDK support for the NIC used. Therefore an Intel I350-T4 with four interfaces is used as NIC. The additional NIC supports 1 GBit/s on each interface.

Our packet generator runs Linux Mint 20.3 Cinnamon with the 5.4.0-109-generic kernel. The specific kernel information is essential because it results in implementation differences in the kernel modules. For automatic packet generation and evaluation, we use the open source packet generator *MoonGen* [44]. „MoonGen is a fully scriptable high-speed packet generator build on DPDK and LuaJIT“ [44]. *MoonGen* supports the saturation up to 10GbE with 64-byte packets while also

executing user-provided Lua scripts for each packet [44]. *MoonGen* allows us to write own Lua scripts and perform our measurements automatically. This allows us to change our network traffic at runtime dynamically. In addition, we can evaluate the time stamps directly at runtime with *MoonGen* in comparison to other packet generators.

To create realistic network traffic, we need to be able to send different packet sizes, manipulate header fields, and change the data rate, all during packet generation. For example, we want to change the destination port from one packet to the next or change the size but keep the same destination. These packet manipulations take place at runtime.

To ensure that we always run the measurements for the same amount of time, we add a command line option to set the measurement duration in seconds. In our measurement setup, the only traffic received by the DUT comes from our packet generator. Hence, the ingress queue of the DUT is empty at the beginning of each measurement, with consequences for latency and packet loss. The first few packets will have lower latency and will not be affected by packet loss, but the following packets will. We add a five-second warm-up phase to each measurement run to exclude this anomaly from our measurement. After that warm-up phase, the actual measurement takes place. During the warm-up phase, we start generating packets but do not measure the latency or packet loss.

Our *MoonGen* script allows us to extract the times tamps from the packets during the receiving process of our packet generator. The receiving process collects the packets, extracts the time stamps, and calculates the measured latency according to Equation (5.3). To increase the performance of our script, we write the calculated latencies periodically as blocks to a csv file. With this methodology, we save ourselves from storing all received packets. Only the latency and packet loss information are stored for further modeling. Consequently, this approach saves us time and space on the hard disk. Thanks to the sufficient available computing capacity and RAM, our performance is not affected by this. Up to the maximum connection speed of the NIC, this method has no loss of performance.

5.4 Design of Modeling Types

With the knowledge of the influencing parameters, we now have to find a suitable modeling type. To create the most accurate modeling possible, we investigate different types of modeling. We look at the advantages and disadvantages of the different types. Based on these results, we choose the modeling type for our further latency and packet loss model.

5.4.1 Modeling with Regression

One method of predicting the value of a variable based on values of other variables is linear regression. The idea behind it is that linear regression tries to find a line that fits the data best. Since our data does not distribute linearly, we test the Gaussian processes regression (GPR). The idea behind GPR is having the regressor to find the best function that fits our data. With regression, we can enter all measured

parameters into the regression. This gives us a complete picture of the performance of the DUT. Table 5.3 provides an overview of the advantages and disadvantages of modeling with GPR.

Pro	Contra
All measurable factors can be taken into account	A lot of test data needed
GPR finds the best function for the given data	Overfitting is likely
	Calculation takes longer than curve fit modeling

Table 5.3: Pros and cons for modeling with GPR

To find the ideal function for the behavior of the DUT, the GPR needs a lot of test data. Ideally, this should be data from tests with every possible configuration of the device and the total network traffic that can occur. This is a considerable effort and not realistic for general modeling in our work. With all the test data, overfitting of the GPR is very likely. Furthermore, the resulting model is only valid for the DUT used. The resulting function reproduces exactly its behavior. Since we want to predict the behavior of software firewalls, we do not desire to overfit. Even the slightest hardware differences make the model unusable. This is why we do not consider modeling with the help of GPR.

5.4.2 Modeling with Curve Fit

Another way of modeling for us is to select the appropriate functions ourselves. We find the functions by analyzing the latency and packet loss measurement data in more detail. For adjusting the functions, we use the curve fit function of the Python package SciPy. This allows us to fit a function based on our measurement results. Table 5.4 lists the advantages and disadvantages of modeling using curve fit.

Pro	Contra
Different functions can be fitted	Variance between the function and some test data can be large
Less test data needed	
Quick model calculation	

Table 5.4: Pros and cons for modeling with curve fit

In contrast to GPR, curve fit enables the usage of several functions for prediction. For certain cluster formations in the measurement results, we can use different functions for each cluster.

We use the fitted function to predict future performance. To get a meaningful function, we need characteristic data of the DUT. In other words, we measure specific settings of the DUT. We use the term setting to refer to various device configurations and network scenarios. This kind of profiling provides us with the

results necessary for modeling. For this type of modeling, we need fewer measurement points than for the regression method. This not only saves us measurement time but also time for calculating the model.

Based on the advantages that curve fit offers us, we use it for our modeling.

5.5 Model Description

In this section, we describe which factors are essential for our modeling. To do this, we use the parameters of the dependency tree in Figure 5.1. We explain which parameters describe the inner state of the device. Then, we formulate the network traffic parameters. Finally, we describe how we measure these parameters.

Our model includes the internal state of the DUT and the current network traffic. We divide the internal state of the DUT into known and unknown parameters. We use the known parameters for our model to predict future behavior. Known parameters, for example, are link speed and firewall rule configuration. Not all parameters are obtainable on every DUT. Therefore, we cannot use all known parameters in our model. These known parameters include the CPU utilization.

The CPU utilization describes the current state of a DUT quite well, but not all DUTs allow us to know this inner state. Moreover, this parameter is hardware-dependent. On a modern CPU, 70 % utilization might not be the same performance as 70 % on an old CPU. For adjusting a behavior model to a DUT more precisely, this parameter can be useful. However, in order to use this parameter correctly, the CPU utilization needs a different unit. Since our model aims to predict more than just one DUT, we do not use the CPU load for modeling.

The situation is similar for the ambient temperature and CPU temperature. We do not know it directly, nor is it available on every DUT. Moreover, it does not matter for our model, as we show in Section 6.4.

To obtain the most accurate prediction possible, it is essential to store the history of internal parameters. The important internal parameters include the firewall rule configuration, the link speed of the device, and the data rate at which packets arrive. Saving this data is essential for our model, as the current state of the DUT has an impact on future behavior.

In this section, we considered parameters that directly influence our traffic model. Next, we consider other parameters that the DUT not directly influences in our traffic model.

5.5.1 Traffic Model

The second important part of our model consists of information about the current network traffic. Since we generate network traffic ourselves in our measurement setup, we have to determine what the network traffic should look like. To do this, we define some framework conditions that our network fulfills:

- The packet sizes are Poisson distributed to generate realistic traffic
- The data rate does not have to be constant

- The average position at which the packets match the firewall rules does not have to be constant, but known

Based on this framework and the dependency tree, we need a minimum number of measurement parameters for modeling. Values we need for our modeling are:

- Packet size range
- Data rate
- Occurred packet loss
- Average firewall rule matching position

These points are the information we need to know about the network traffic in order to define our model. Packet size and data rate are the determining parameters for the network traffic behavior. By using information about packet loss, we can improve the description and modeling of our DUT. Other parameters are not necessary, such as the selected packet transport protocol. This is due to the different sizes of the headers for TCP and UDP. The TCP header is slightly larger than a UDP header. The firewall only checks the headers. The slight difference in size between the two header types has almost no influence on the performance of the DUT. Apart from the header size, the firewall processes the packets in the same way.

Our network contains parameters that are not taken into account or not allowed. For this purpose, we define not allowed network states. The resulting network state parameters are not part of our traffic model. We do not permit the following aspects in our traffic model:

- A packet size distribution other than the Poisson distribution
- Firewall features such as Deep Packet Inspection (DPI), SSL Inspection, Anti Spyware, etc.

The Poisson distribution serves as a realistic packet size distribution. We can map special networks, e.g., only small packets or rather large ones. In addition, a Poisson distribution also maps the Simple IMIX. We discuss the reasons for this in Section 6.1.5.

Additional features are not allowed in our traffic model because they are only available on a subset of firewalls (NGFWs). Options such as DPI are part of an Intrusion Detection System (IDS) and therefore not exclusively a firewall feature. In our model, we only consider firewall features.

The parameters we generate with our traffic model allow us to specify the model for the DUT. To measure the parameters as accurately and uniformly as possible, we define our measurement model.

5.5.2 Measurement Model

In our measurement model, we describe the different parts that we record during a latency measurement. We also show which network changes we pay attention to.

A single measurement point contains the data rate, packet size, packet loss, average firewall rule matching position, and the measurable internal state. The duration of a single measurement run is two seconds. The two seconds of recording start after a five-second warm-up period. We do this to make sure that the first packets measured do not cause outliers, as they may end up in an empty firewall queue. Several of these measurement points together make up one measurement run. A measurement run without any DUT configuration provides the basic results for our modeling function. Over a time interval t, we store several intermediate states, e.g., measurement points. We are storing these values results in our measurement history. The history is necessary to track increases and decreases of individual parameters during a measurement run. With the help of the model history, we can determine the future behavior of the DUT.

In our measurement model, we allow changing the data rate and the position of the matching firewall rule during runtime. Due to the Poisson distribution, the sizes of the incoming packets are constantly changing. We do not change the measurement duration because two seconds are a good balance between the number of packets needed to calculate an accurate picture of the DUT behavior and the overall measurement duration.

The parameter packet loss is essential in the sense that we not only use it for our model but also indicate the quality of the traffic. Depending on the application, different packet loss rates are tolerable. J. Mwela et al. [45] state that the quality of video streams can tolerate a packet loss of up to 1 %. R. Pauliks et al. [46] state that 0.25 % of IP packet loss is acceptable for video streaming. One of the reasons why these protocols tolerate packet loss is because it happens. This shows us that it is essential to consider packet loss in our model. It is not only part of the modeling but is also used to investigate use cases. When selecting devices, we consider the degree to which packet loss is tolerated by the application. Based on this, we can select network devices.

5.6 Profiling Steps

For a precise prediction model, the quality and quantity of the input data are essential. To achieve the best result with a minimum effort, we create an automatic profiling script. The script contains several measurement configurations that our model requires. In this section, we describe the necessary profiling steps.

All measurements consider Poisson distributed packet sizes. Our script varies the maximum and minimum limits of the Poisson distribution to test multiple network scenarios. By varying the packet sizes, we ensure that we also test the Simple IMIX distributed packet range.

The first measurement we perform in our script is a baseline measurement. We configure no rules on the DUT. This ensures that we can measure the base latency of the DUT. With this setting, we also measure when the DUT enters an packet

loss behavior. This measurement is essential for the foundations of our model. Our basic measurement proceeds as follows:

- Data rate set to 1 MBit/s with Poisson distributed packet sizes
- Eight data rates between 1 MBit/s and the maximum data rate with Poisson distributes packet sizes
- Maximum data rate with Poisson distributed packet sizes

After that, we reconfigure the firewall with 50 firewall rules. The last rule is the matching rule for our traffic, so we compare 50 rules in any case until the matching rule. With this configuration, we generate the same network traffic again. Thus, we get a first impression of the influence of firewall rules on the performance of our DUT. We perform the same measurement for 100, 400, and 1000 firewall rules. With these measurement results, we get a good impression of the behavior with firewall rule configuration. Why we use exactly this measurement point for our profiling is explained in Section 6.2.1.

To find out more properties of the DUT, our script measures how the DUT behaves under load changes. To do so, we configure the firewall in the same way as in our baseline measurement. The difference now is the generation of our network traffic. After one second of measurement recording, we increase the data rate by 100 MBit/s during the measurement run. We simulate a random load change of the network traffic. The resulting data helps us to adapt our model to real network traffic.

We also measure the effect of firewall rule position changes during the measurement run. After one second of the measurement recording, we change the average firewall rule matching position from 50 to 1.5. With this, we test the DUT for its behavior under rule configuration changes.

To simulate particular network traffic, we can specify our packet sizes even more precisely. If rather small packets appear in a network, we take this into account in our profiling. Therefore we start the profiling with a different network traffic setup. With our profiling, it is possible to do the same for large packets. We generate the corresponding packet sizes in our packet generator by setting the minimum and maximum values of the Poisson distribution. This option allows us to obtain an even finer understanding of the DUT, which is useful for simulating our model.

5.7 Simulation Model

The last step in the design chapter is the simulation of our model. Simulating our model offers various possibilities. Through simulation, we test our model without hardware. With little effort, we gain a better understanding of the system through simulation. This also supports possible system decisions. Since we also model the influence of system changes, the simulation helps develop a strategy. With little effort, we make it possible to capture the system complexity and map the temporal progression. The simulation offers an alternative to real experiments with which we can test arbitrary scaling.

In addition to the opportunities, the simulation also has some risks. The simulation requires a high initial design effort. The design defines what the simulation condition should look like and its properties. It is essential to ensure that the construction is accurate. Otherwise, it will falsify the simulation of our model. Another risk of simulation is incorrect modeling. These include the following errors:

- Simplified assumption
- Unrealistic model
- Lack of transparency
- Logical error
- Lack of data
- Garbage-in-garbage-out system

For us, it is essential to prevent these errors during modeling. Otherwise, we risk a defective model without additional benefit. Apart from that, a simple programming error can cause an incorrect simulation. Numeric errors or the randomness of the result are also risks of a simulation. The apparent objectivity of the model makes the user believe the model would work. But it is only applicable to this simulation. To prevent this error, it is essential not to treat the simulation as ground truth. In our simulation, we consider the risks mentioned above to avoid them.

In our simulation, it is essential to test the model for its reliability and accuracy. For this purpose, we design the simulation in such a way that various network scenarios are testable. The results provide information about the behavior of the DUT. With this information, we are able to make decisions about the usability of devices in specific network scenarios.

6 Implementation

In this chapter, we discuss the implementation of our model and introduce the core properties.

The performance of the firewall depends on several parameters. In Chapter 5, we describe the framework conditions for our network and the parameters whose influence on firewall performance we are investigating. The parameters we investigate, from Section 5.2, are the transport protocol, data rate, packet size, the distribution of packet sizes, cross traffic, base load, number of firewall rules, the position of the matching firewall rule, the firewall rule type, and the temperature. If we combine all these parameters into one prediction function, we create an unnecessarily complex prediction model. Therefore, we look at the behavior of the parameters and try to find correlations among them. Based on this information, we can combine the parameters.

The data rate, packet size, and distribution of packet sizes investigations are part of Section 6.1. Which transport protocol we use in our network traffic is part of our discussion in Section 6.1.4. In Section 6.1.7, we describe the cross traffic parameter in more detail. We examine the influence of the firewall rules, their position, and type in Section 6.2. The base load is part of our CPU measurement in Section 6.3. In Section 6.4, we take a closer look at the influence of temperature on the performance of firewalls. Finally, we create our model and describe the various functions.

6.1 Traffic

In this section, we examine the influence of the network traffic parameters from Section 5.2. First, we consider the data rate, the packet size, and their effect on the network performance. Second, we discuss the parameter packets per second (pps). Third, we compare the behavior of TCP and UDP on the performance of firewalls. Next, we discuss the distribution of the packet sizes for realistic network traffic generation. Afterward, we consider the variation of the data rate. Finally, we define what impact cross traffic has on the firewalls. We perform the measurements on the EAGLE40 with iptables (DUT 1), see Table 5.2, for a definition of the devices.

6.1.1 Date Rate

An important parameter describing the network traffic is the data rate. By keeping all other parameters constant and varying only the data rate, we determine its influence on the latency of the DUT.

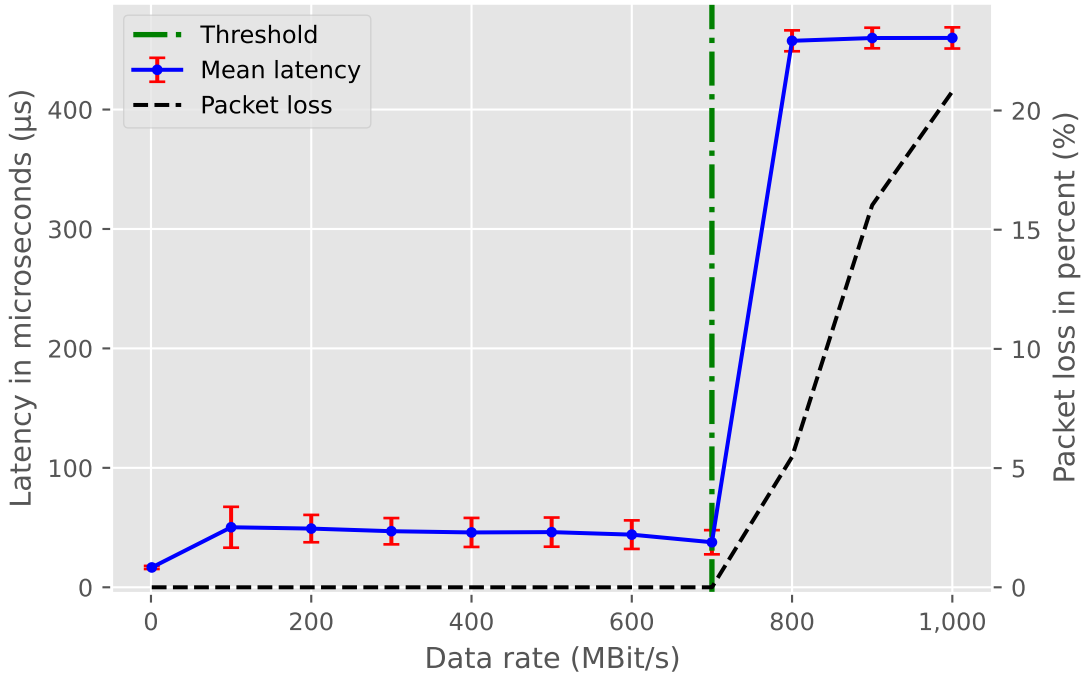


Figure 6.1: Measured mean latency in microseconds and packet loss for different data rates with constant packet size of 160 Bytes; Recording for 2 seconds at each measurement point; Depending on the firewall configuration the position of the threshold changes; We define left side of threshold as *non-overload area* and right side as *overload area*

Figure 6.1 shows an example of how the data rate influences the latency and packet loss. We measure the latency behavior for a constant packet size of 160 Bytes. The DUT 1 does not contain firewall rule configurations in this measurement. Each measurement point is the mean latency from 2 seconds of recording. Two seconds of recording are enough because, before that, we already generate five seconds of traffic without recording it. In the five seconds before, we eliminate possible anomalies at the beginning of the transmission. The error bars indicate the standard deviation of the latency measurement. Figure 6.1 demonstrates how the change of the data rate leads to a change in latency and packet loss. At data rates below 500 MBit/s, the measured latency is below 100 μs, and the packet loss is 0 %. Above 500 MBit/s, the latency increases to over 400 μs, and the packet loss rises to 21 %. The variation in latency (jitter) is larger in the range below 500 MBit/s than in the range above since the firewall has reached its overload area. We define the term *overload area* as the area where the firewall starts to suffer packet loss and the latency increases. A firewall in an industrial network must not operate in the overload area, as they do not tolerate packet loss. In Figure 6.1, we show the threshold through the green dotted line. The threshold changes depending on the firewall rule configuration and is not always at the same position. On the right side of the threshold is the overload

area where packet loss occurs. We define the term *non-overload area* as the left side of the threshold. In the non-overload area, there is no packet loss, and we observe lower latencies in this range. Since the firewall can no longer process all packets and generates a high latency for the packets, the variation in latency decreases.

With Figure 6.1, we show that the latency of the DUT increases with the data rate. The higher the data rate, the higher the mean latency. Accordingly, the data rate has a direct influence on latency behavior. As soon as the firewall reaches its overload area, the packet loss starts to increase. The packet loss then also increases with the data rate.

6.1.2 Packet Size

The second important network traffic parameter whose influence on latency we investigate is the packet size. How we generate this is defined in Section 6.1.5. To determine the influence of the packet size on the latency behavior of the DUT, we change the packet size from measurement to measurement and not within a single measurement recording.

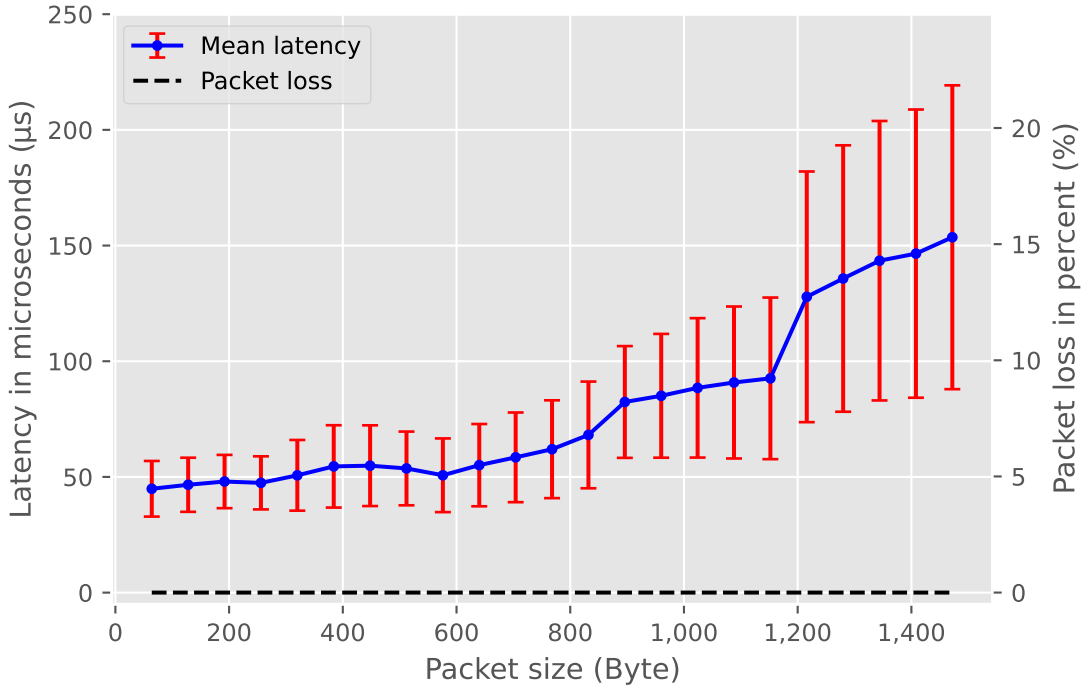


Figure 6.2: Measured mean latency in microseconds for different packet sizes with constant 200 MBit/s data rate; Record for 2 seconds at each measurement point

In Figure 6.2, we show how the latency of the DUT 1 behaves when changing the packet size. At a constant 200 MBit/s, the latency increases with increasing packet

size. At 64 Bytes, the mean latency is less than 50 μs, rising to over 150 μs at 1522 Bytes. The packet loss remains constant at 0 % over the entire packet size range.

The increasing latency is due to the physical size of the packet. Within the DUT the forwarding of the larger packet takes longer than that of a smaller one. Accordingly, the packet size affects the latency of the DUT. The packet loss remains constant at zero, since 200 MBit/s does not lead to an overload of the firewall in any packet size combination.

From Figure 6.1 and Figure 6.2, we conclude that both parameters influences the latency and packet loss of an DUT. In Section 6.1.3, we show how the combination of the two parameters affects the latency behavior and what conclusions we can make from this.

6.1.3 Packets per Second

Figure 5.1, shows that many parameters influence the latency of a firewall. The increasing number of parameters leads to the increased complexity of our model. This is because our model has to consider many parameters. If we look at how the data rate and packet size affect the latency of the DUT, we observe that both influence the latency.

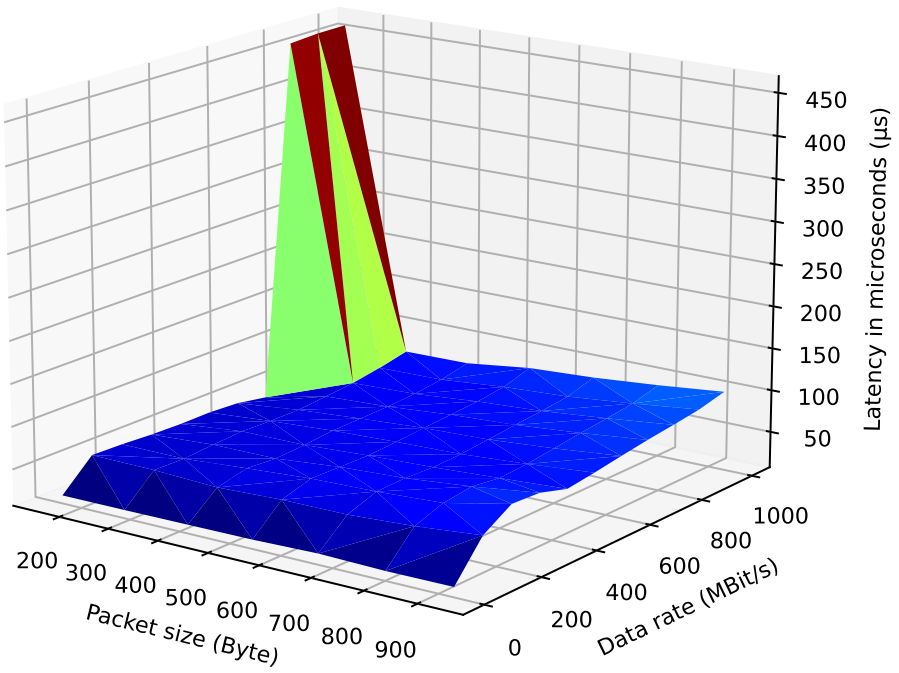


Figure 6.3: 3D model of DUT 1 base measurement

In Figure 6.3, we create a three-dimensional surface to show the latency influence of packet size and data rate. We generate the data for the 3D model through a *base measurement*. In a *base measurement*, we only vary the packet size and the data rate. For the *base measurement*, we do not configure any firewall rules or other options on the DUT.

The majority of the area in Figure 6.3 (shown in blue) represents a latency below 150 μs. Small packet sizes of 160 bytes at 1 MBit/s have a mean latency of less than 50 μs. The latency increases continuously to over 100 μs for packets with a size of 928 Bytes at 1000 MBit/s. We show that larger packet sizes results in higher latency. However, from 800 MBit/s to 1000 MBit/s, we observe that the small packet sizes have a higher latency than large packets at the same data rate. This area is the overload area. In Figure 6.3, we display the overload area as a dark red and green surface. The overload area displays that the firewall needs too long to process the packets. As we show in Figure 6.1, the overload area is related to the packet loss. In the *base measurement*, in Figure 6.3, are too many small packets that the firewall has to forward. As a result, the firewall generates packet loss and has increased latency. In this case, a latency of over 450 μs.

With a data rate of 1000 MBit/s, we observe that the latency temporarily decreases from 160 byte to 288 byte. Since the firewall is no longer in its overload area the latency decreases. Starting from 288 Bytes the latency increases again with the packet size. At the same data rate, we send fewer large packets than small ones. The firewall is then again able to process the packets without packet loss. The larger packet sizes prevent the firewall from entering its overload area.

To prevent our model from becoming unnecessarily complex due to too many influencing parameters, we combine the data rate and the packet size into one parameter. The resulting pps parameter allows easier comparison of the DUT. We show how to calculate the parameter pps in Equation (6.1). To calculate the pps, we divide the data rate r in MBit/s by 8 to obtain the data rate in MByte/s. We divide the data rate in MByte/s by the packet size s in Bytes. To the given packet size we add 20 Bytes for the preamble, start frame delimiter, and interpacket gap. We show the layout of the layer 1 Ethernet frame in Figure 5.3.

$$pps = \frac{\frac{r}{8}}{s + 20\ Bytes} \tag{6.1}$$

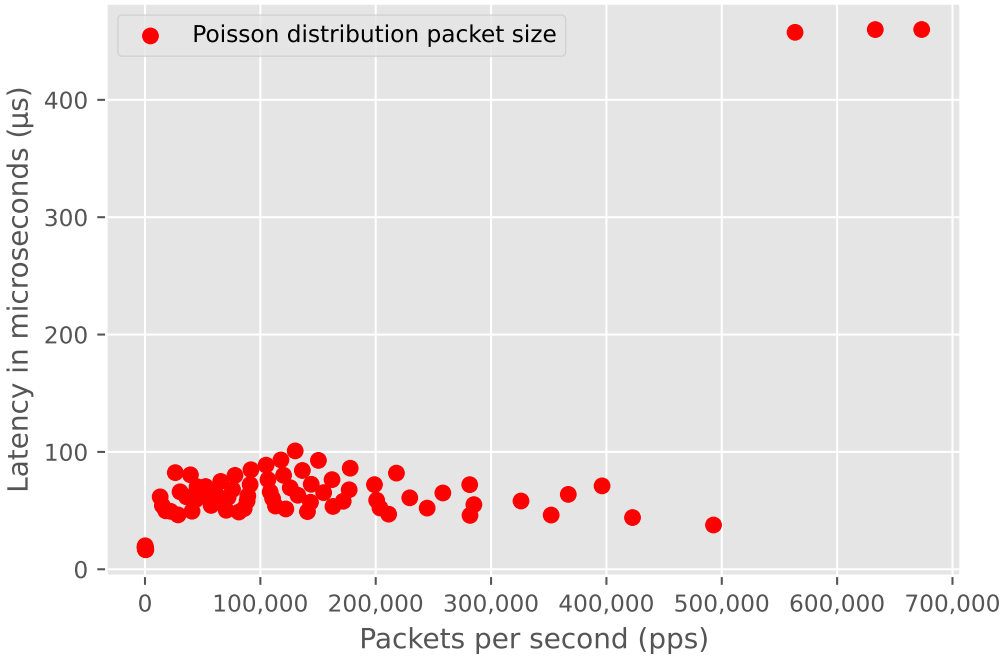


Figure 6.4: DUT 1 base latency measurement in μs with no firewall rules configured

With this combination of data rate and packet size, we simplify our model. In Figure 6.4, we display and compare different latency measurement results in pps. Using the pps parameter, we can display latency measurements and packet loss measurements across multiple data rates and packet size combinations in 2D images. Each point in Figure 6.4 represents the latency in microseconds for a pps value. Most points up to $500,000$ pps have a latency of less than 100 μs. The overload area starts at about $550,000$ pps and the latency increases to over 400 μs. The 2D representation with pps not only simplifies the presentation of measurement results. With the pps parameter, the prediction functions are only dependent on one parameter instead of two.

However, the presentation does not only have advantages. Due to the combination of packet size and data rate in pps, it is no longer obvious from which network setting the measurement results to originate. We observe that in the range below $500,000$ pps, the points have a spread of about 50 μs. One function will not be sufficient to map all latencies.

6.1.4 Used Protocol

In our system model (Section 5.1), we define that we support both TCP and UDP as transport protocols in our model. In this section, we examine the differences between the two protocols for our model and their influence on the measurement results. A TCP header is slightly larger than a UDP header. We examine how crucial this size difference is since a firewall looks at the header of a packet when checking them. For the generation of the TCP traffic, we use the Ostinato packet

generator instead of MoonGen, because it offers the possibility to generate stateless TCP traffic. Unfortunately, a correct TCP connection is not possible with either of the two generators.

In Figure 6.5 we compare the latencies of UDP and TCP traffic. We represent the standard deviations in the TCP an UDP latency measurement by error bars. For our comparison, we measured three pps values with TCP packets that represent both the overload area and the non-overload area. Our TCP measurement at 200,000 pps has an average latency of 48.7 μs and a standard deviation of 11.4 μs. In comparison, the UDP measurement with the same number of pps has an average latency of 58.9 μs and a standard deviation of 8.1 μs. The three TCP measuring points show a similar pattern as the UDP measuring points. Therefore, we decided to measure only these three pps values with TCP packets.

We can see that the latencies of the TCP traffic behave similarly to those of the UDP traffic. The reason for the similar latency is the processing of the DUT. In our packet generation, both TCP and UDP packets are the same size. At the same packet size, the size of the payload is different for TCP and UDP. A packet inspection firewall processes TCP and UDP packets in the same way. The additional TCP header fields are not relevant for the firewall. Since the TCP packets have only a slightly larger header than the UDP packets, the latency difference is small.

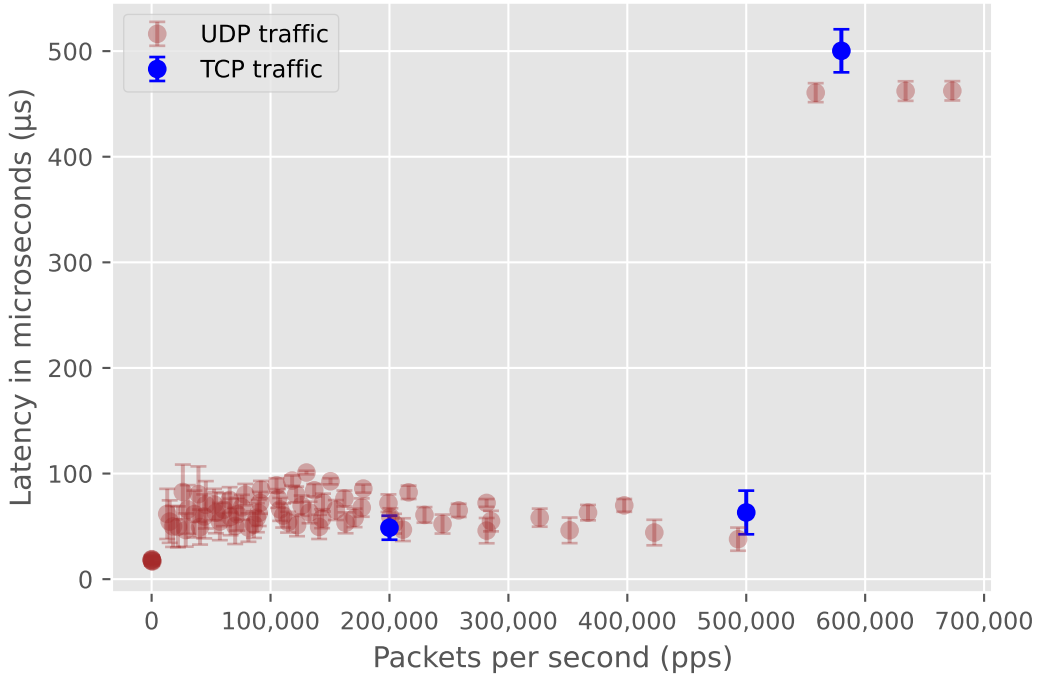


Figure 6.5: Comparison of the latency measurement between UDP and TCP traffic

Due to the small latency difference, the transport protocol we use for our network traffic generation does not matter. Therefore, we use UDP packets for the network traffic generation in our measurements.

6.1.5 Packet Size Distribution Function

In order to obtain good latency and packet loss measurement data for the creation of this prediction model, we need realistic network traffic. To make our network traffic realistic, we need a distribution function that defines our packet sizes. To find a suitable distribution function for the packet sizes, we use the Simple IMIX as a basic distribution. Our distribution function for realistic packet sizes must at least be able to reproduce the Simple IMIX distribution. If possible, we find a distribution function that helps us to map an even larger pps range.

The generation of packet sizes must be pseudo-random, as we need to know the size of the packets sent at any given time. Otherwise, there is no guarantee that the packets arrive at the specified data rate.

As an approach to generating realistic packet sizes, we first generate static packet sizes. MoonGen allows the generation of multiple packets during runtime. This allows us to generate packets with static size according to the specifications of the Simple IMIX and send them according to the Simple IMIX sequence. The biggest disadvantage of this is the lack of dynamics in generation, as Simple IMIX only defines three different packet sizes. If we test with this distribution up to a maximum data rate of, e.g., 1GBit/s, we only achieve $\approx 346,904$ *pps*. With only 64 Bytes packets, however, up to $\approx 1,488,095$ *pps* would be possible.

To make the network traffic more realistic, we need a distribution function that generates the packet sizes pseudo-randomly at runtime. For an efficient and simple generation of such sizes, the Poisson distribution is a good choice. This is shown in Figure 6.6. The figure shows the similarities between Poisson distributed packet sizes and Simple IMIX distributed ones. Each point in the plot represents the average latency of a measurement. We define one measurement by a data rate and the maximum and minimum packet size for the Poisson distribution. Figure 6.6 also shows the range of our Poisson distributed network traffic. From 1 *pps* to $694,000$ *pps*, almost all combinations are possible.

The latency of the Simple IMIX distribution changes only marginally. This is because Simple IMIX has fixed packet sizes and a fixed packet generation order. Therefore, the latency does not change dramatically.

We can generate the Simple IMIX distribution by using the static packet size definition in MoonGen. For the Poisson distributed packet sizes we change the packet size at runtime. For this purpose MoonGen offers the possibility to generate Poisson distributed values. We can assign these values as sizes during the creation of the packets. After we assigned the random size to the packet, MoonGen transmits it.

The Poisson distribution does not only represent the Simple IMIX distribution but also can generate larger pps combinations. Thus, the Poisson distribution fulfills our requirements and we can use it for the creation of our model.

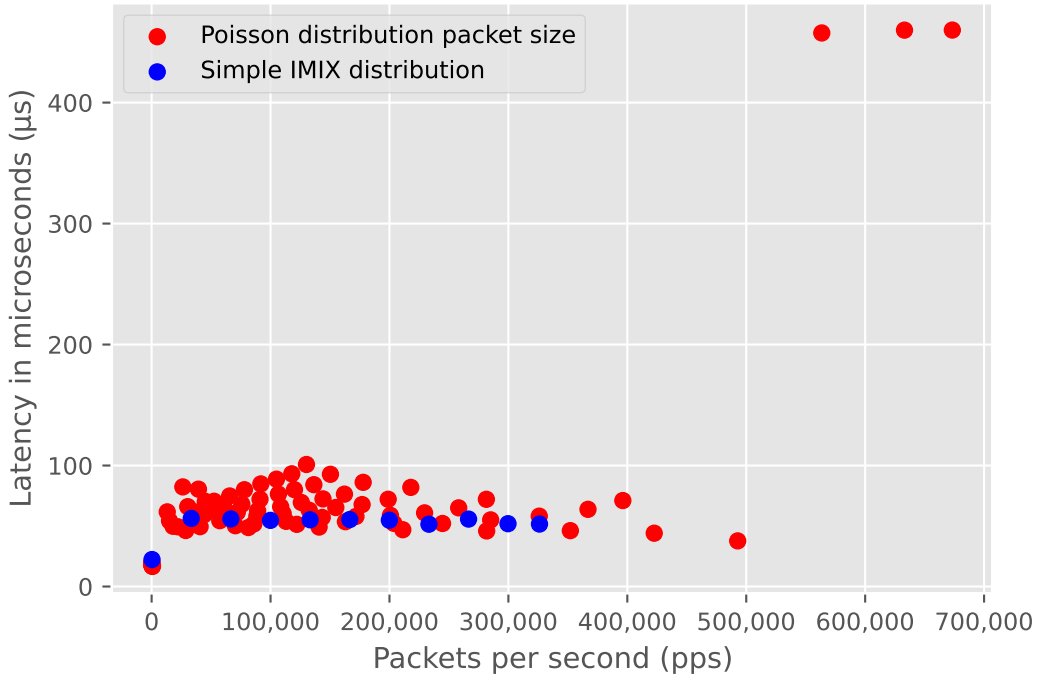


Figure 6.6: Comparison of a measurement with Simple IMIX distributed packet sizes and a measurement with Poisson distributed packet sizes

6.1.6 Variation

In the environment of a network, load changes occur in random events. In a corporate network, such a load change may occur by triggering a security camera. If the security camera detects a person, the camera uploads its video recording to a server. This increases the data rate on the network. Another example of data rate alternation in a network is the transfer of machine data. In a production environment, machines transfer data about a work-piece between each other. The data contains, for example, information for future processing steps. Sending the work-piece data from one machine to another increases the data rate in the network for a short time. In this section, we investigate the effects of data rate changes during runtime.

During a measurement run, we change the data rate by 100 MBit/s and observe the effects. A change of 100 MBit/s corresponds to a rapid increase in the data rate. Such a change in the network is triggered, for example, by a backup routine. In this scenario, the backup routine uploads data to a server at a certain time interval. By uploading the data, the data rate in the network increases for a short time.

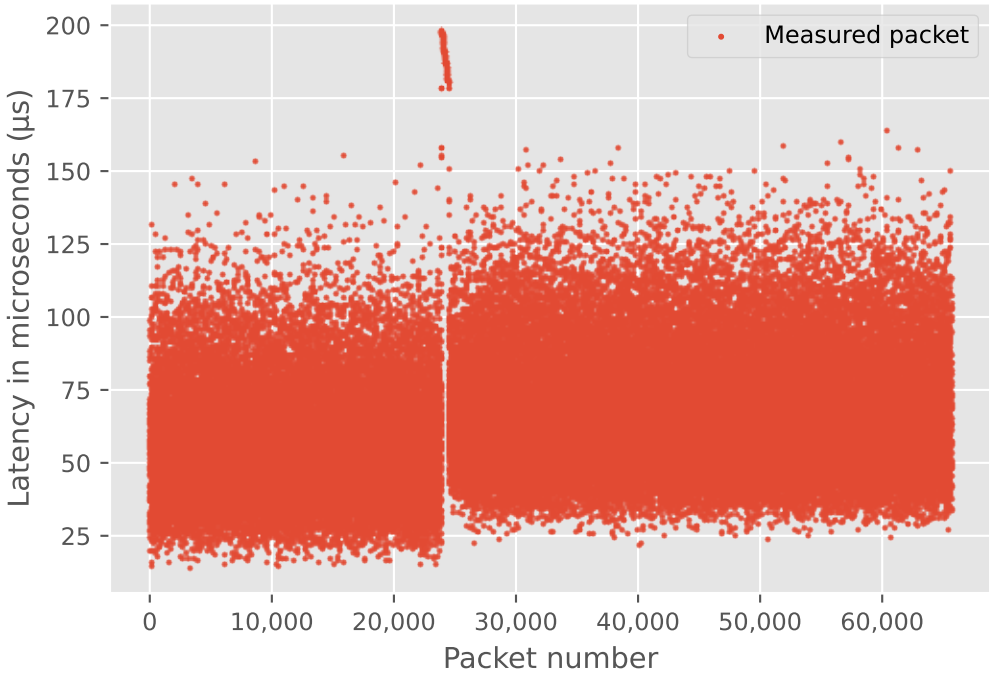


Figure 6.7: DUT 1 measurement run with data rate change after one second

We present the result of such a measurement run in Figure 6.7. Each measurement point in Figure 6.7 represents the latency of a packet. The load generator sends packets with Poisson distributed packet sizes between 64 Bytes and 1024 Bytes. We record a total of 2 seconds. At the beginning of the measurement, we send with 100 MBit/s. After one second we increase the data rate to 200 MBit/s. We maintain the increased data rate for the remaining second. The spike does not occur in the middle of the graph because we display the packet latencies in the order of their arrival. In the first second, we send 22,163 packets. After the data rate increase, we send a total of 44,326 packets. That is why we are counting more packets after the data rate increase.

The sudden increase in data rate results in a latency spike. We can observe the latency spike at around packet number 22,200. The latency spike occurs after we generate the first 22,163 packets at 100 MBit/s. Only with the knowledge about the network status at time $t-1$ we can make a statement about the behavior at time t. Therefore, we have to store the network history in our model.

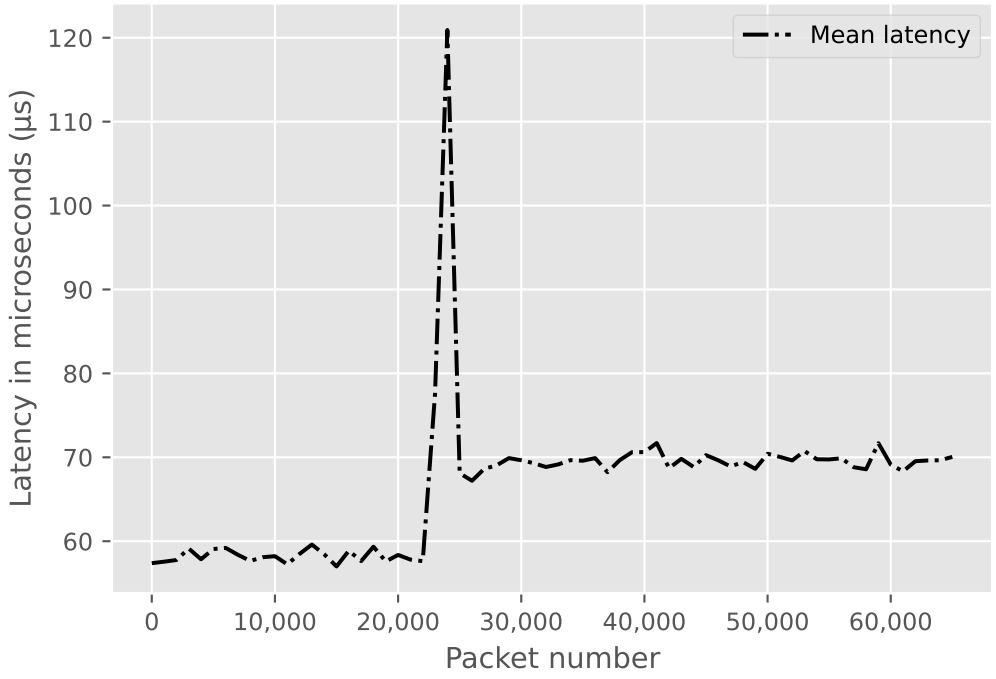


Figure 6.8: Mean latency curve of the DUT 1 data rate change measurement

Since we only preserve the spike in Figure 6.7 and not the average latency change, we show this separately in Figure 6.8. The line we show in Figure 6.8 represents the mean latency of the data rate change measurement run. We observe that after the data rate changes, the latency remains at a higher level than before. This is what we expect, as we already show in Figure 6.1 that higher data rates, result in higher latency values.

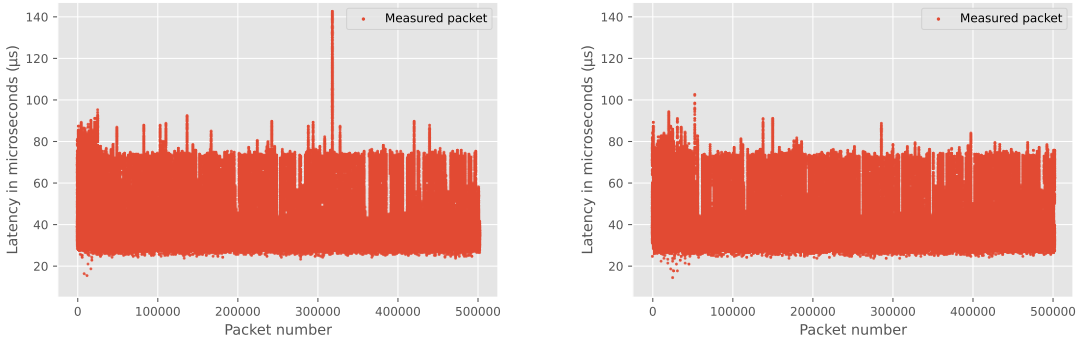
6.1.7 Influence of Cross Traffic

Our goal is to design the latency and packet loss model for real network conditions. During our investigation of the latency and packet loss behavior of the DUTs, we accessed the DUTs via SSH. We use the access to start several scripts that record and save the desired CPU parameters locally on the DUT. After the measurement, we transfer this data to the packet generator via SSH.

Part of a real network scenario is the generation of additional load on the DUT. We generate this load, for example, by access to the web interface or configuration via SSH. With the SSH connection generate cross traffic on the DUT.

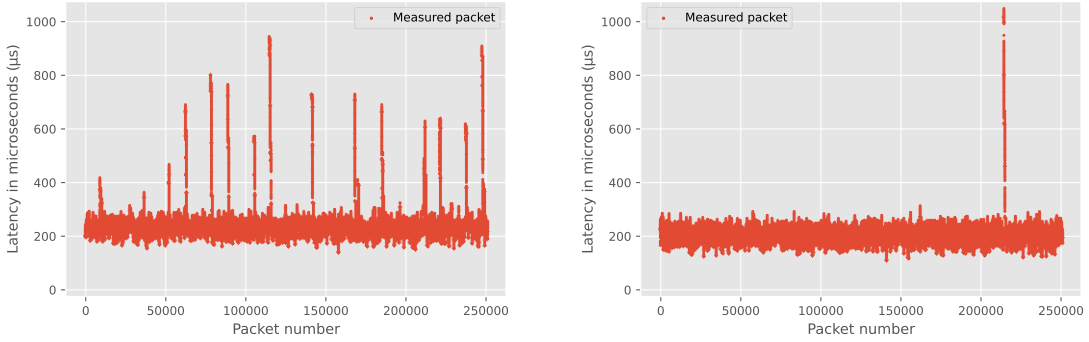
In real applications, the DUT can be accessed at any time via a web interface, SSH or monitoring tool. The problem with cross traffic is that we cannot simply predict it. No parameter tells us when the access will occur. In order to determine the impact on our latency and packet loss model, we need to determine how much the cross traffic affects the latency and packet loss.

We compare the latencies with and without cross traffic in Figure 6.9. Figure 6.9(a), shows the latency behavior of the DUT 1 with SSH connection. In Figure 6.9(b), we display the latency behavior of the DUT 1 without SSH connection. We perform both measurements at 700 MBit/s, the Poisson distribution generates packet sizes between 64 and 256 Bytes, and we configure no firewall rules on the DUT. We can observe that the latency experiences more upward swings with cross traffic. These outliers are due to the SSH traffic. As soon as the SSH connection transmits data, the data rate and the composition of the packet sizes change for a short time. We cannot accurately determine the latency and frequency of these outliers. This is because we do not consider the exact composition of SSH traffic in our modeling.



(a) With SSH connection (b) Without SSH connection

Figure 6.9: Latency behavior on the DUT 1 with/without cross traffic measurement



(a) With SSH connection (b) Without SSH connection

Figure 6.10: Latency behavior on the DUT 1 with/without SSH connection

In another latency measurement, shown in Figure 6.10(a), we show the influence of cross traffic on firewalls where we configure firewall rules. We measure the latencies at 800 MBit/s. Our Poisson distribution generates packet sizes between 300 and 1024 Bytes. In addition, we configure 100 firewall rules. Due to the configured

rules, the cross traffic generates a significantly higher latency spike. The DUT takes even more time to process the packets due to the cross traffic.

In Figure 6.10(b) we show the same measurement but this time without the SSH connection. We see that the random outliers disappear except for one. Latency spikes can occur during measurements but by preventing cross traffic, we reduce these outliers. System interrupts, for example, can cause such outliers.

The latency outliers influence the behavior of a DUT. However, as we observe in Figure 6.10(b), a worst case can occur without cross traffic. The latency of outliers influences the mean latency behavior of the DUT. By generating cross traffic on the DUT during our profiling measurements, we train the latency and packet loss model with this behavior. This provides us with a latency and packet loss model that also works in real network environments. Without cross traffic in the profiling measurements, our latency and packet loss model would predict too low mean latencies.

6.2 Firewall

In this section, we examine the influence of firewall rules on the DUT performance. First, we analyze the influence of firewall rule configurations on latency behavior. Second, we describe the behavior when packets match at different firewall rule positions. The third step is to show the effect of changing the firewall position during the measurement recording. Finally, we show the influence on the performance that connection tracking has. We perform the measurements for this purpose on the DUT 1, see Table 5.2, for a definition of the devices.

6.2.1 Firewall Rule Configuration

In this section, we examine the effect of firewall rules on latency behavior. The number of firewall rules plays a crucial role in the latency behavior of a firewall. To determine the firewall rule-dependent latency, several measuring points are necessary. We have exact firewall configurations that we test. Before we start the measurements, we configure the number and order of firewall rules on the DUT using a SSH script that we implemented. The measuring points we determine characterize the behavior of a firewall. Using these measurements, we model the behavior of the firewall. Our measurement takes place on the EAGLE40 with standard Ubuntu 20.04.3 LTS, we use standard iptables (DUT 1).

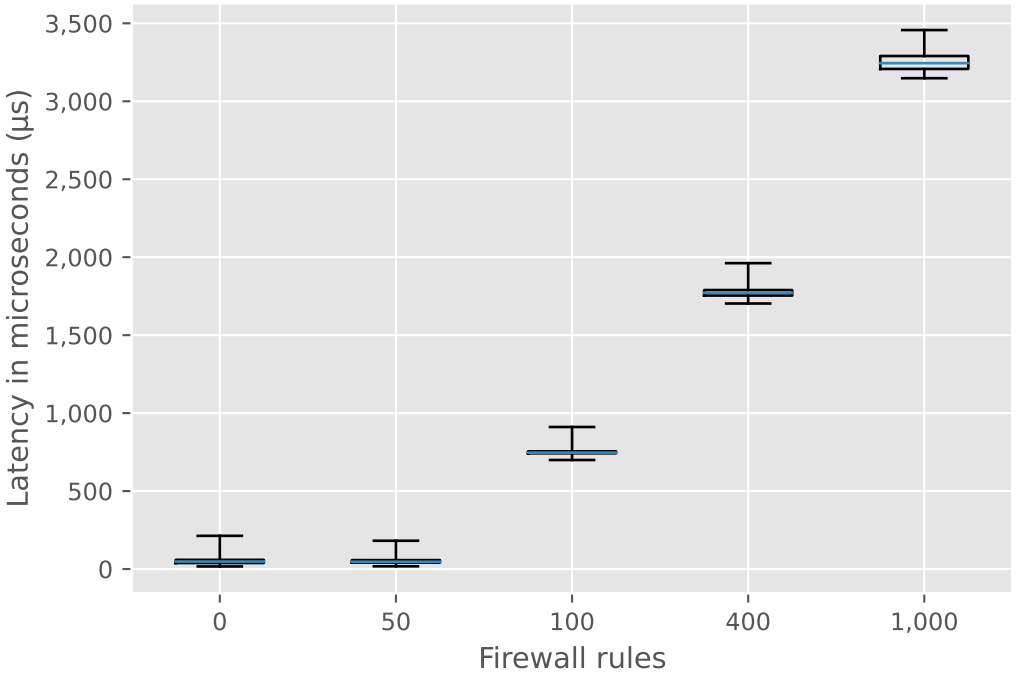


Figure 6.11: Measured mean latency for different firewall rule matching positions; 160 Bytes packet size and 500 MBit/s data rate

Figure 6.11 shows how the latency behavior of the DUT 1 changes with the configuration of the firewall rules. We draw the box plot boxes from the first to the third quartile. The horizontal line within the boxes denotes the mean latency. In Figure 6.11, the whiskers show the maximum and minimum latency that we observe. In this measurement, the packet size is constant at 160 Bytes, and the data rate is fixed at 500 MBit/s. The only parameter we change is the position of the matching firewall rule. We measured 0 | 50 | 100 | 400 | 1000 configured firewall rules. The more firewall rules we configure before the matching rule, the higher the latency is. The position where packets match in the firewall filter table influences the firewall overload area. Due to the matching position, the overload area already starts at less pps and the latency increases. Therefore, this parameter also plays a decisive role in modeling a firewall. Based on these observations and our definition of the overload area from Section 6.1.3, we analyze how the firewall rule configuration affects the overload area.

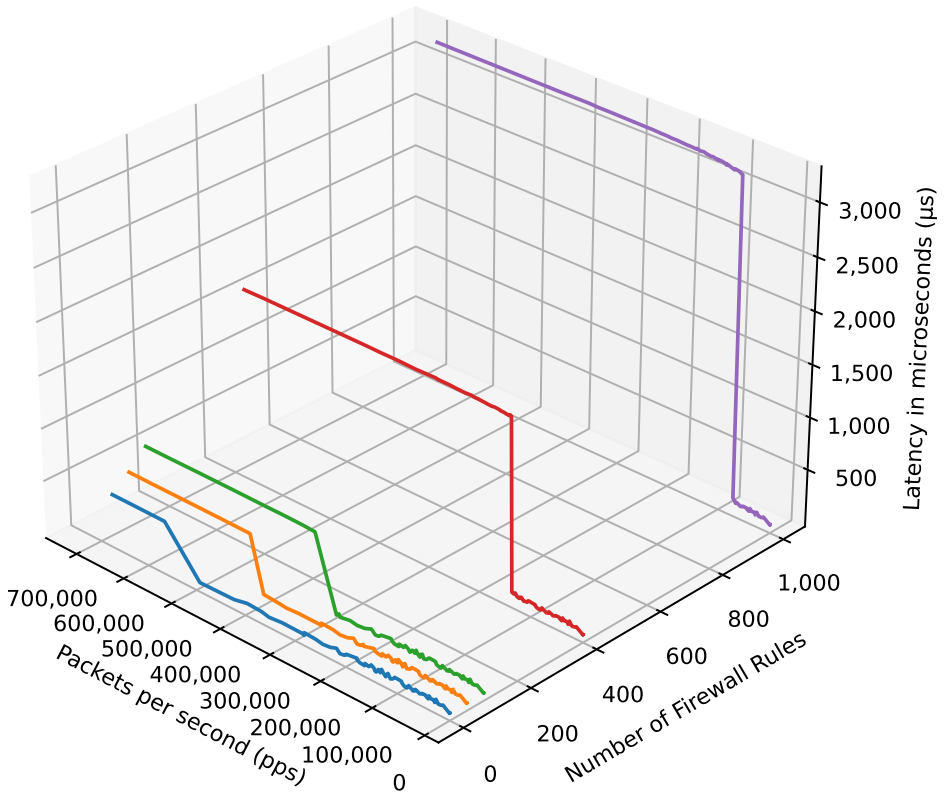


Figure 6.12: Comparison of firewall rule configuration measurements with 0 | 50 | 100 | 400 | 1000 firewall rules

In Figure 6.12, we compare the latency curves of all firewall rule configuration measurements. Each line represents one latency curve for the corresponding number of firewall rules. The blue line shows the latency behavior without firewall rules. From about 500,000 pps, the latency increases significantly, and the firewall reaches its overload area. The red line, which represents the latency behavior with 400 firewall rules, already shows this at 160,000 pps. The more firewall rules we configure, the earlier the latency starts to increase, and the higher the maximum latency rises. We call this point at which the latency starts to increase *threshold*. The overload threshold is the point at which the DUT changes from the non-overload area to the overload area.

We consider the threshold points separately in Figure 6.13. The Figure 6.13 shows the number of pps before the firewall enters the overload area. This pps value represents our overload threshold. It is noticeable that the threshold decreases with increasing firewall rule matching position. This means the more firewall rules we configure, the closer the threshold of the overload area approaches 1 *pps*. We explain this behavior by the fact that the firewall checks each rule before the matching rule. The more rules we configure beforehand, the more rules our firewall checks until the first one applies. The more rules the firewall checks, the more time it takes. With iptables, we can configure the firewall rules in such a way that packet loss already

occurs with the first incoming packets. The latency is accordingly immediately on the overload area level. With this knowledge and the measurement results from Figure 6.13, we deduce that the position of the applied firewall rule affects the threshold of the overload area exponentially. The exponential function approaches 1 *pps* asymptotically.

The dashed curve is the result of fitting an exponential function with the five measured thresholds and the Python SciPy curve fit function. We use the Python SciPy curve fit function to fit the data we observe to an exponential function. The result has such a high deviation, compared to Figure 6.14, because too few points are available for the curve fit function. To overcome this problem, we interpolate linearly between the points to generate fifty additional points for the fitting function. How we deal with the increased measurement error due to the interpolation is described in Section 6.5.9.

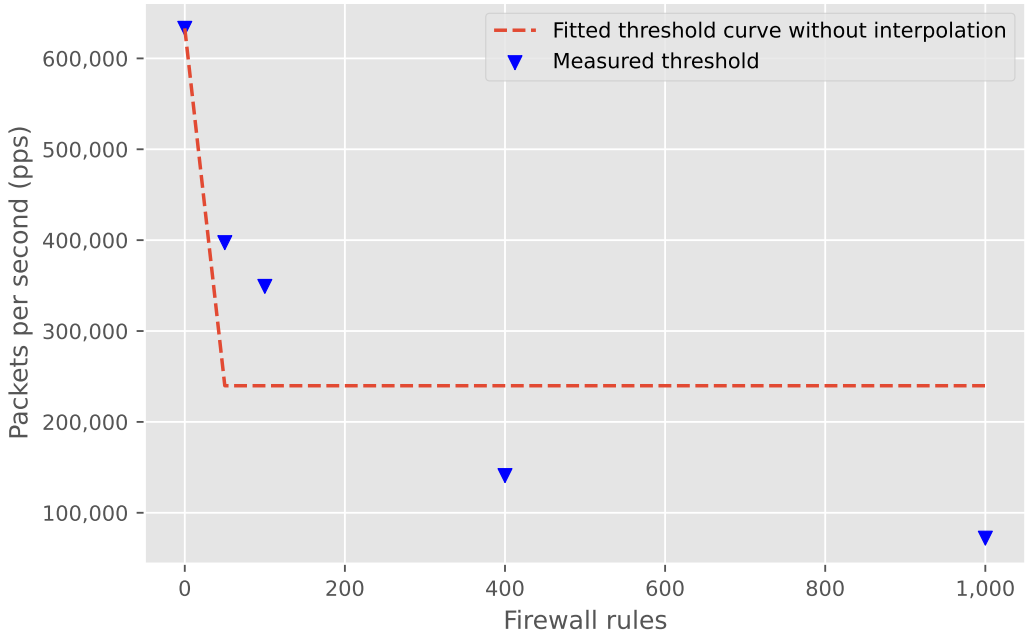


Figure 6.13: Firewall rule configuration measurement and threshold curve without optimization

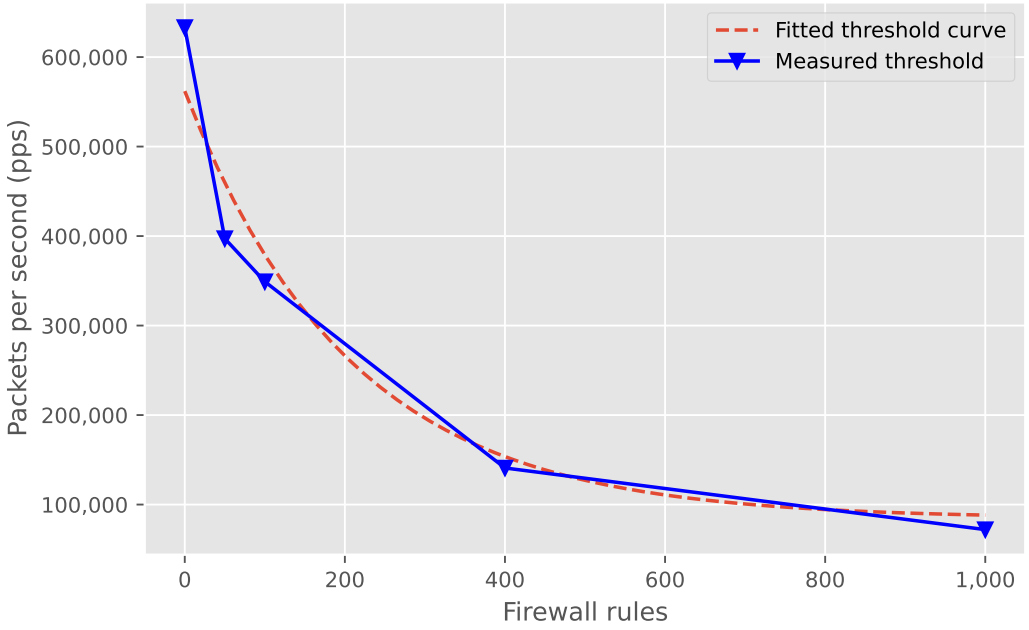


Figure 6.14: Firewall rule dependent threshold in pps with linear interpolation between measured points

With the help of interpolation, we get a much better result in terms of fitting the exponential function to the points. In Figure 6.14 we show the optimized fitted curve. We interpolate linearly between the measured thresholds. In the figure, we connect the points linear. The dashed line represents the result from our curve fitting. The deviation between the measurement points and our adjusted exponential function is 6.25%. Before our optimization (Figure 6.13), the deviation was 74.82 %.

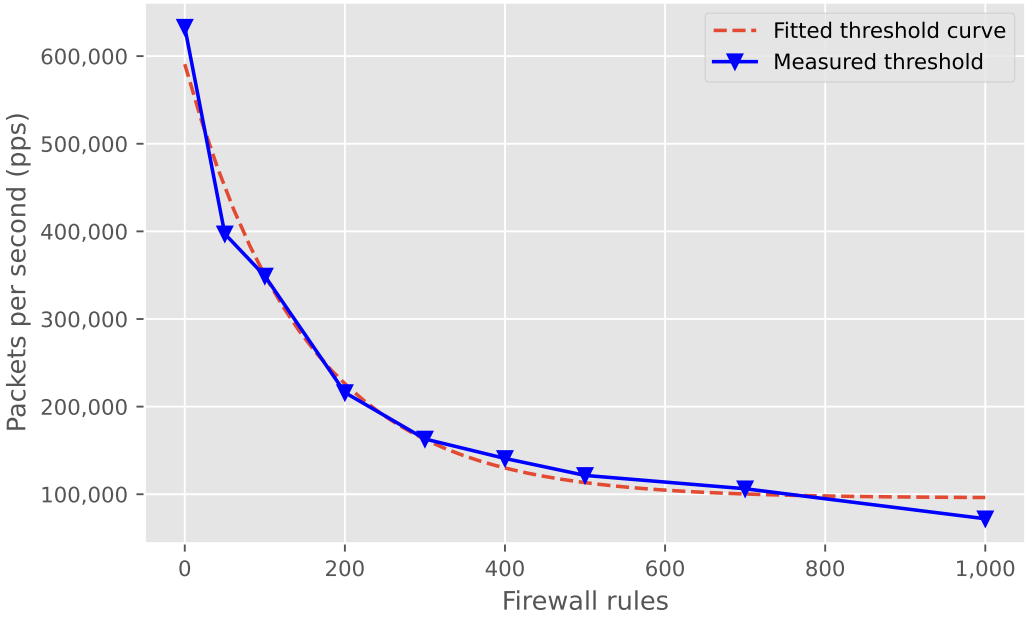


Figure 6.15: Firewall rule dependent threshold in pps with more measured points; Compare with Figure 6.14

As soon as we include more measuring points in our test, the measurement takes longer, but we do not get a better fitting result. For illustration purposes, we show the result in Figure 6.15. We measure the same firewall with nine firewall rule configurations instead of five. We observe that the measurement results decrease linearly between 200 and 500 firewall rules. Four additional firewall rule configurations increase the measurement time by approximately 80%. Nevertheless, the deviation between the measurement points and the adjusted exponential function is 7.66%. With a linear interpolation between 200 and 500 firewall rules, we would observe the same result and save ourselves two firewall rule configuration measurements.

We derive the overload threshold function from the measurement data of our firewall matching rules. Based on this overload threshold function, we predict the overload threshold value. Equation (6.2) is the exponential function that we use for fitting our threshold behavior. The variable n determines the average firewall matching position. The overload threshold function asymptotically approaches 1 pps. There cannot be an overload threshold at zero or a negative pps value. We define that the overload threshold function is only valid in the positive value range.

$$g(n) = a \cdot e^{(-b \cdot n)} + c \tag{6.2}$$

$$g(n) \in \mathbb{R}^+ \tag{6.3}$$

$$n \in \mathbb{R}^+ \tag{6.4}$$

Based on the calculated exponential function, we determine the firewall-dependent overload threshold. The overload threshold function represents a part of the firewall

rule influence in our model. As already mentioned, the number of rules that the firewall needs to check influences the latency. To predict the latency influence, we use the firewall matching rule measurement we already did. From the measured values, we average the latencies in the overload area and the non-overload area. By doing this for each measurement configuration, we obtain several points that we use to predict the latency behavior. Figure 6.16 shows the averaged latencies for the overload area and the curve fitted to the firewall matching rule measurements. The measurement results show us that the latency increases linearly from 459.91 μs with 0 firewall rules to $3,155.56$ μs with 1000 firewall rules. Our function that we fit to the measuring points is thus also linearly increasing. We explain the measured latency behavior by the way the filter table works. The firewall checks for every packet any rule in the filter table until it finds a match. The more rules we configure before the applicable rule, the longer this comparison takes. Since the number of rules is limited only by the used hardware, the latency increases linearly with the number of configured rules.

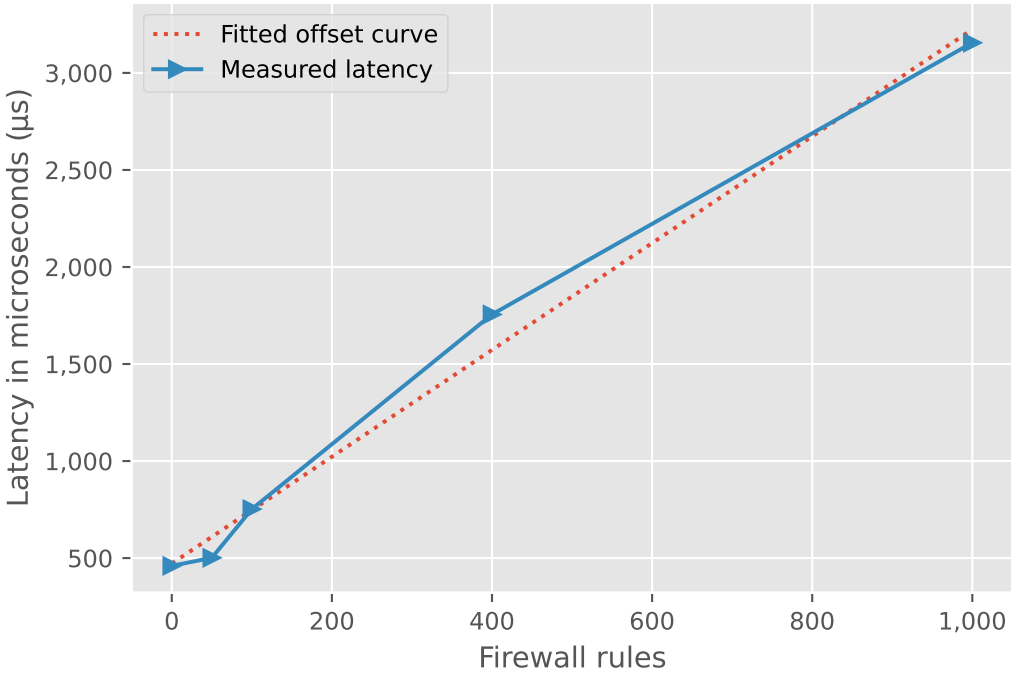


Figure 6.16: Firewall rule dependent latency offset for the overload area

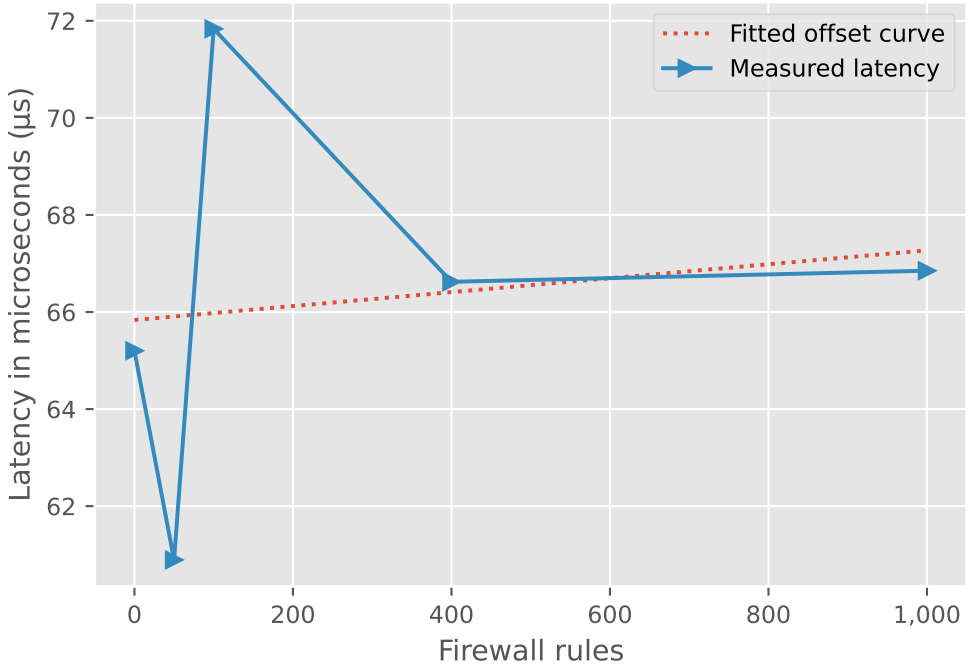


Figure 6.17: Firewall rule dependent latency offset for non-overload behavior

For the non-overload area, the latency increases with an increasing number of rules, but not monotonously. Figure 6.17 shows this behavior. We show in Figure 6.17 that the latency in the non-overload area varies and increases with an increasing number of firewall rules. With 0 firewall rules we measure 33 μs this decreases to 26.6 μs with 50 firewall rules and with 100 firewall rules the latency increases to 37.8 μs. In comparison to the overload area in Figure 6.16, the slope of our fitted function is not so large.

In Figure 6.18 we show the latency curve for several measuring points. Figure 6.18 shows how differently the latencies behave in the non-overload area. The average firewall rule matching position does not affect the non-overload area as much as the overload area. Nevertheless, an increased matching position leads to higher latency in the non-overload area of the firewall. Normally, we would expect the latency to increase with the increasing number of firewall rules. However, there are some deviations in our measurements where the latency drops unexpectedly, as we observe in Figure 6.17. The deviations of the latencies upwards and downwards are due to measurement errors. In Section 6.5.9 we describe how we deal with these measurement errors.

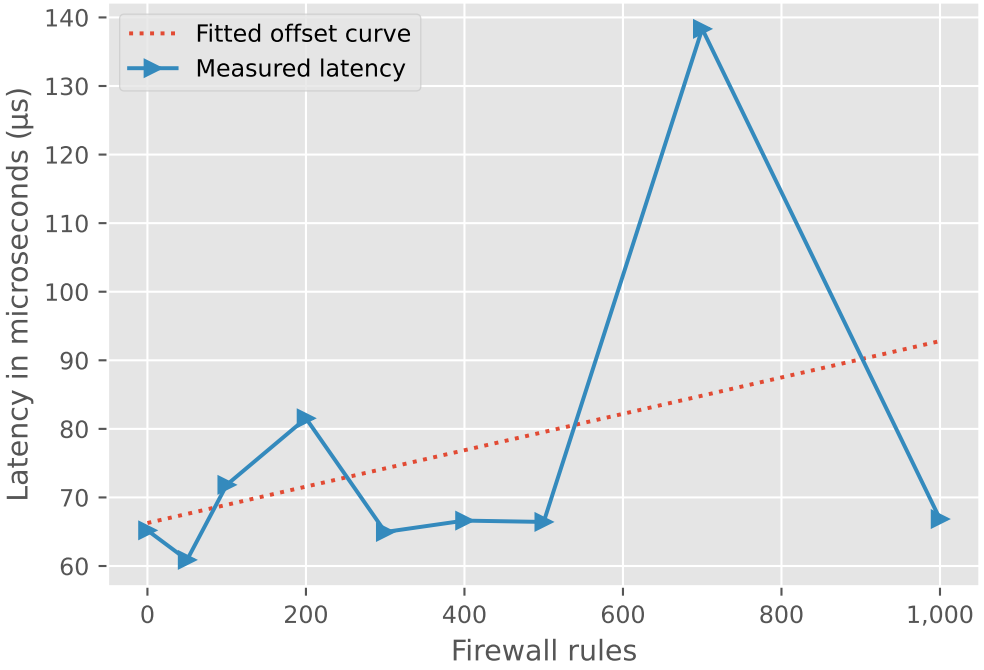


Figure 6.18: Firewall rule dependent latency offset for non-overload behavior with nine firewall rule configurations

From our discoveries about the latency offset shift, we create the following function:

$$h(n) = \begin{cases} i \cdot n + j & p \geq g(n) \\ d \cdot n + m & p < g(n) \end{cases} \tag{6.5}$$

$$h(n) \in \mathbb{R}^{+} \tag{6.6}$$

In Equation (6.5), $h(n)$ determines the latency offset that the average matching position of the firewall specifies. In our model, we select the function depending on the pps value, p. If p is greater than or equal to $g(n)$, we use the overload offset function to determine the latency. The knowledge we collected on the firewall rule matching position dependency is useful for our prediction model.

6.2.2 Average Firewall Rule Position

In a real network scenario, all packets do not match only one firewall rule, because the packets have different source and destination addresses. Thus, in an example configuration with 100 firewall rules, 25 % of the packets match at position 20 and 75 % match at position 40. This difference also affects the latency, since the packets that match at position 20 have a smaller latency than those at position 40. We solve this problem by specifying the average firewall rule matching position. In the preceding example, the packet distribution results in an average matching position

of 35. The entire traffic then behaves as if all packets match at position 35. We demonstrate this behavior with an example in Figure 6.19.

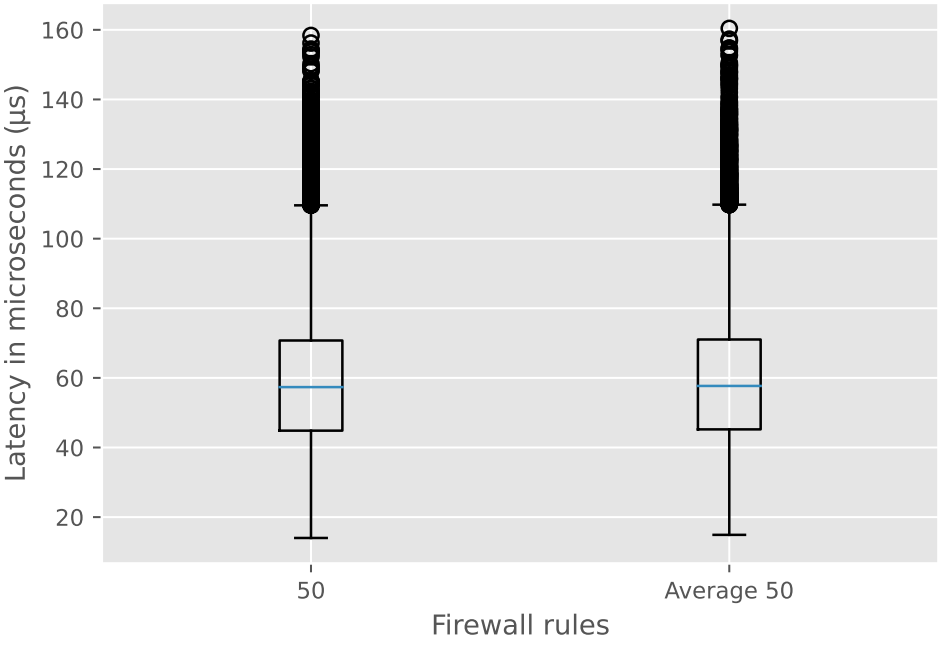


Figure 6.19: Latency in microseconds; Comparison between firewall rule matching position 50 and average firewall rule matching position 50; 100 MBit/s data rate and Poisson distributed packet sizes between 64 Bytes and 1024 Bytes

In Figure 6.19, we show the latencies of two measurements. We draw the box plot boxes from the first quartile to the third quartile. The horizontal line within the boxes denotes the median latency. The whiskers span 1.5 times the inter-quartile range (IQR) from the box. Points drawn outside the whiskers are outliers. The box-plot with the label 50 corresponds to a measurement in which all packets match at the firewall rule position 50. In this measurement we observe a mean latency of 59.28 μs. The other box-plot corresponds to a measurement where 50 % of all packets match at position 1 and the rest matches at position 99. Our measurement with the average matching position 50 results in a mean latency of 59.55 μs. Despite the different matching positions, the latencies and jitter behave approximately the same. In our model, we therefore consider the average matching position.

6.2.3 Firewall Rule Matching Position Change

In another measurement, we observe what happens by changing the position of the firewall rule matching position. We measure what happens when we move the average matching position from 1 to 2000 during the measurement run. Because we want to provoke the firewall directly into the overload behavior, we choose the matching position change so large. In this measurement, we generate the packets at

a data rate of 100 MBit/s. The Poisson distribution generates packet sizes between 64 and 1024 Bytes. In Figure 6.20, we plot every packet latency we measure as a red point. The black dash-dot line represents the mean latency of all packet latencies in this measurement run. If we look at the result of this measurement in Figure 6.20, we notice that there are no spikes in the measurement. In contrast to the data rate increase, the change of the firewall rule matching position does not lead to any latency spike. After we change the firewall rule matching position from 1 to 2000, the latency increases from an average of $\approx 59\ \mu s$ to $\approx 82\ \mu s$. This corresponds to the behavior we describe in Section 6.2.1. The latency remains at the higher latency level after the matching position change. This means for our modeling that we have no additional effort concerning the rule change.

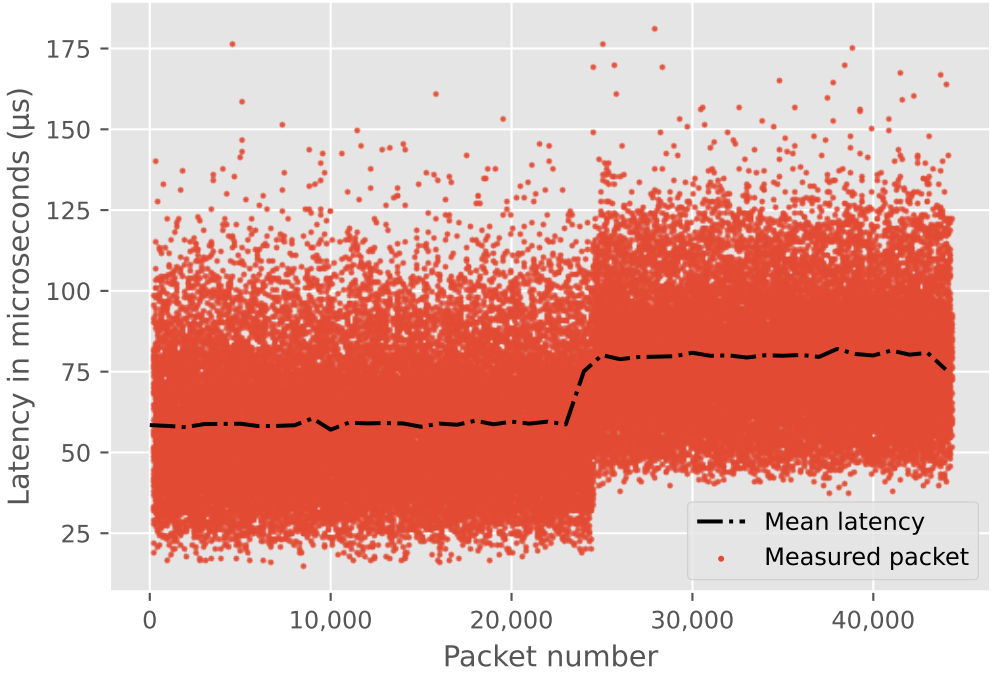


Figure 6.20: DUT 1 measurement run with firewall stateless rule matching position change after one second form 1 to 2000; Constant 100 MBit/s data rate; Poisson distributed packet sizes between 64-1024 Bytes

6.2.4 Influence of Connection Tracking

So far, we did not look at the influence of connection tracking on the behavior of the firewall. In this chapter, we will examine connection tracking. Firewalls allow connection tracking with the configuration of stateful rules. The implementations differ between the various software firewalls. Connection tracking is an essential part of many firewalls. Many networks use connection tracking, hence its implementation needs to be efficient. Below we discuss the implementation of connection tracking.

First, we examine the connection tracking functionality of iptables. Second, we discuss how VPP implements connection tracking.

6.2.4.1 Iptables Connection Tracking

Iptables is a kernel module of Netfilter and is only used to communicate with Netfilter in the Linux kernel [47]. This means that Netfilter controls the connection tracking. To control the connections, Netfilter uses the *conntrack* module. The Netfilter kernel loads this module automatically on demand [14]. The exact *conntrack* implementation depends on the respective kernel. We show the general packet flow in Figure 6.21.

Every incoming packet first passes through the Prerouting hook [14]. This calls the *conntrack* function and checks if the packet is one of the following possible things:

- It is part of or related to a tracked connections.
- The first packet of a not yet tracked connection.
- An invalid packet.
- Marked as NOTRACK.

If the first packet is already part of an existing connection, the *conntrack* system looks in a central hash table to find the possible match. The packet then receives a pointer to the corresponding entry. This is necessary for further kernel modules. In our example, we assume that we forward the packet. Accordingly, it passes through the forward hook. The packet must also pass through the forward filter chain, despite the already existing pointer. If iptables finds the appropriate stateful rule, the packet continues sending. The Postrouting hook processes the packet further. In this case, the *conntrack help+confirm* function has no effect [14].

The second point assumes that the packet is the first of a connection we do not track yet. In this case, the packet also passes through the Prerouting hook. As in the previous case, the *conntrack* function searches for a match in the hash table. Since iptables finds no entry this time, the function creates a new struct for the packet. The *conntrack* system still sees the packet as "unconfirmed", therefore it is not yet saved in the central hash table. On its way through the forwarding hook, the forward filter chain checks the packet. After iptables finds the appropriate rule, iptables processes the packet in the Postrouting hook. The last function that the packet traverses before it continues is the *conntrack help+confirm* function. The purpose of this function is to confirm the new connection. To do this, *conntrack* moves the connection instance from the unconfirmed list to the connection table. From this point on, the connection is part of the hash table and *conntrack* tracks the connection. This is because iptables can drop the packet between the first *conntrack* function and the *conntrack help+confirm* function. An example of this: In the forward chain, the administrator defines a rule that drops the corresponding connection. This way *conntrack* ensures that the connection table does not fill unnecessarily.

If we send an invalid packet, *conntrack* will not discard the packet. Conntrack registers it as invalid and sends the packet on. This is because there can also be stateful rules with an INVALID state.

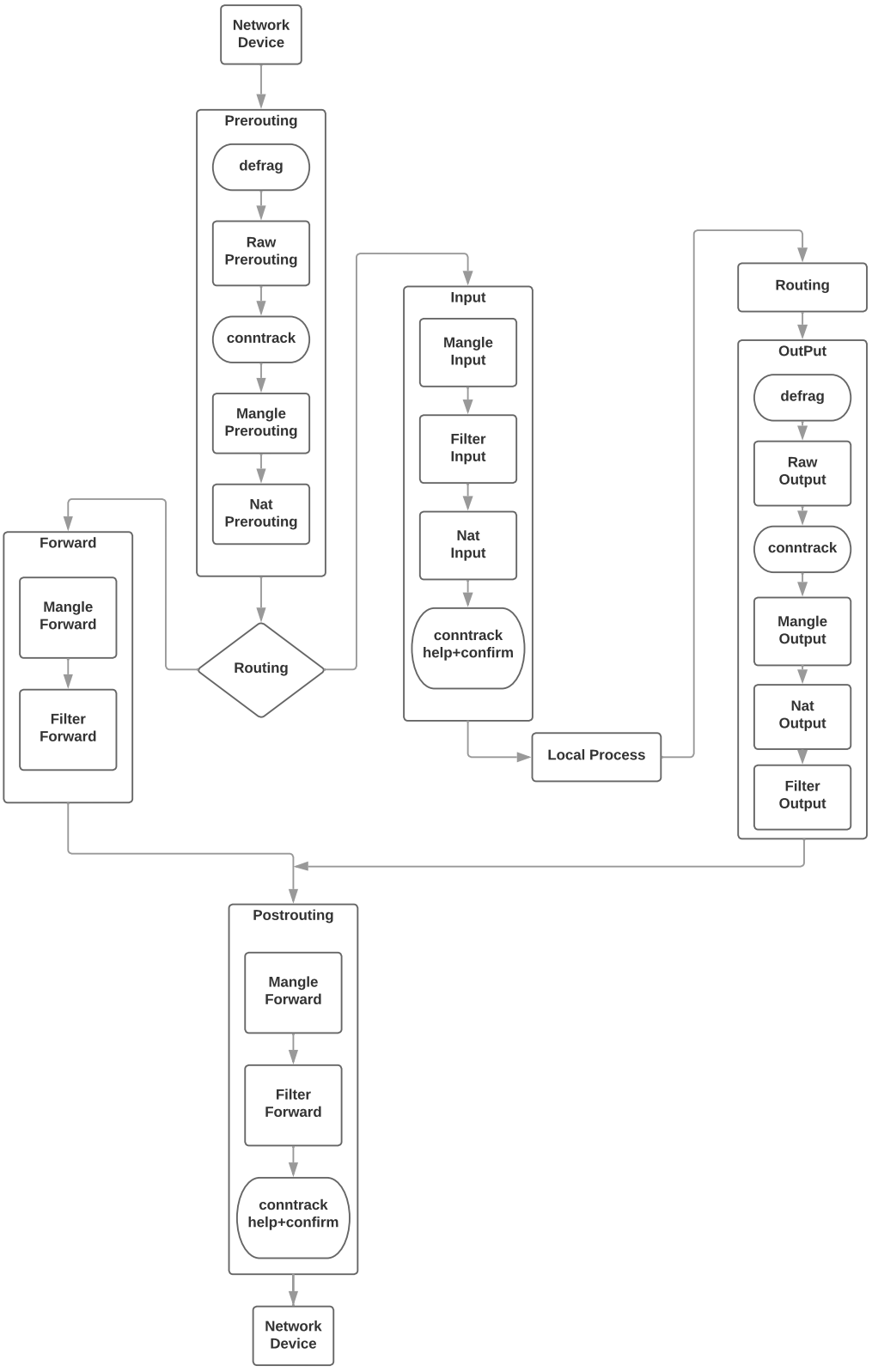


Figure 6.21: Iptables chains

If we mark a packet as NOTRACK, *conntrack* ignores the packet.

In the implementation of iptables connection tracking, every packet traverses the filter chains. Regardless of whether iptables finds an entry in the *conntrack* table. An example of why this behavior is useful: If the administrator configures a stateful rule, *conntrack* generates the corresponding entry in its table after the first packet of this connection. After some time, the administrator reconfigures the firewall. The administrator not only deletes the corresponding stateful rule. Instead, he configures a rule that drops this connection. Without checking the filter table for a match, the following occurs. In this case, every packet that we generate directly after the reconfiguration is not affected by the change. The reconfiguration would only take effect after the *conntrack* timeout expires. This can take more than five days.

Now let's look at the effects of this behavior on our DUT. In Figure 6.22 we show the result of a measurement with configured stateful rules. Our goal in this measurement is to show how the latency behaves when we configure stateful rules. Each red dot represents the latency of a single packet. The black dash-dot line represents the average latency of the packets. In the first second of our measurement, the matching firewall rule is at position one. After one second, the matching stateful rule changes to position 2000. We observe that the mean latency increases from $\approx$ 56 μs to $\approx$ 83 μs. The data rate is constant at 100 MBit/s during the entire measurement. The packet generator sends Poisson distributed packets with 64-1024 Bytes size.

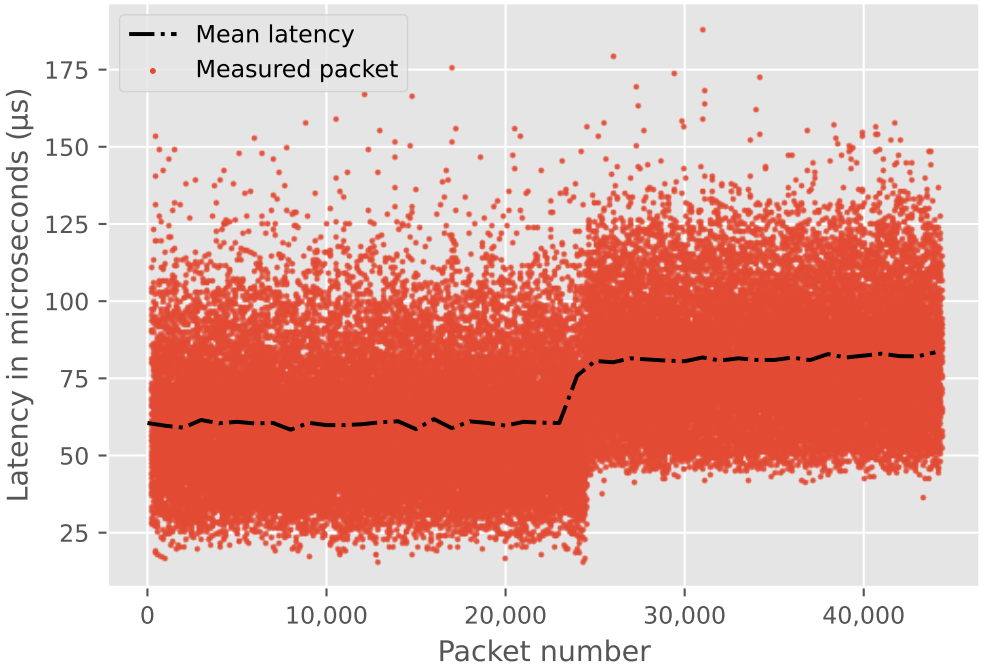


Figure 6.22: DUT 1 measurement run with firewall stateful rule matching position change after one second form 1 to 2000; Constant 100 MBit/s data rate; Poisson distributed packet sizes between 64-1024 Bytes

Note that the latency increases significantly after the rule position change. We show that despite the connection tracking, each packet has to pass through the 2000 rules until it reaches the matching rule. In comparison, we observe the same increase in Figure 6.20. The measurement conditions are the same except for the type of matching rules. In Figure 6.20 we use stateless rules and in Figure 6.22 stateful rules. Since we observe the same behavior, we show that connection tracking with iptables does not reduce latency.

6.2.4.2 VPP Connection Tracking

VPP is another software firewall we are investigating. In this section, we describe how VPP enables connection tracking. We focus on the differences between VPP and the connection tracking implementation of Netfilter.

VPP uses an ACL plugin to support stateful and stateless ACLs. Stateful rules in VPP are ACL rules with the action permit+reflect. Since the stateful rules are the most used, they are heavily optimized [48]. Just like *conntrack* in iptables, VPP uses a hash table to store connections. Unlike *conntrack*, VPP only generates entries if we configure a corresponding ACL permit+reflect rule. Thus, VPP only has to take the overhead of generating the entry once. After that, only one look-up is necessary for each packet of this flow.

For each incoming packet, the VPP firewall looks in its connection table to check if there is an existing entry. If VPP finds a suitable entry, it forwards the packet straight on and does not check the other ACL rules. This approach creates an enormous latency advantage over iptables.

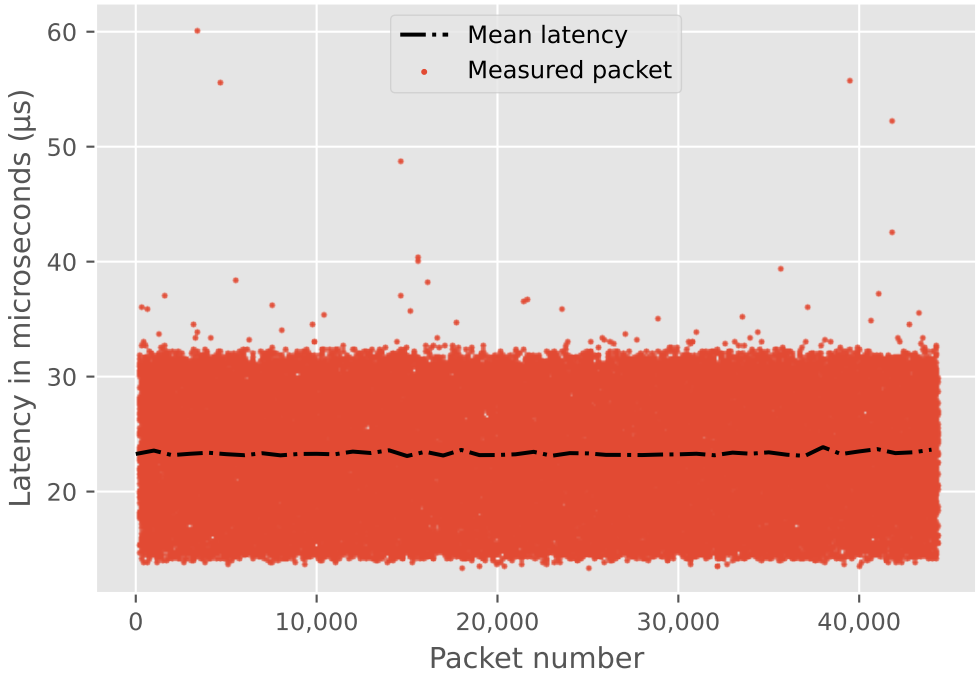


Figure 6.23: DUT 3 measurement run with ACL rule permit+reflect matching position change from 1 to 2000 after one second; Constant 100 MBit/s data rate; Poisson distributed packet sizes between 64-1024 Bytes

Figure 6.23 shows the latency behavior of VPP. As in our measurement in Figure 6.22, we first configure the matching firewall rule at position one. After one second, we send a packet flow that matches a stateful rule at position 2000. With VPP, the matching rule change does not affect the latencies of the packets. The mean latency remains constant at $\approx 23\ \mu s$. This shows that the procedure in VPP offers better performance with stateful rules.

6.3 Central Processing Unit (CPU) Measurement

We determine how the CPU frequency, CPU temperature, and memory usage behave during a measurement. To determine the influence that the hardware has on the performance of the DUT, we measure various CPU parameters. We display the CPU parameters in relation to the data rate. We especially want to examine how the CPU parameters change in the overload behavior. This ensures that the firewall reaches a packet loss scenario.

CPU frequency, CPU temperature, and memory usage can all be indicators of improving or deteriorating performance. First, we examine the CPU frequency. With a constant load, an increase in CPU frequency can lead to faster processing of tasks, and a decrease in frequency can slow down processing. Second, we look at the CPU temperature. In general, temperature affects the performance of all electronic

devices, so we also examine the CPU temperature behavior. The temperature limit behavior is examined more precisely in Section 6.4. Third, we consider memory usage. With an overloaded memory, the DUT is no longer able to process packets. If this occurs during our measurements, it can be a reason for decreasing performance. Finally, we show the effect of our CPU parameters on the packet loss.

6.3.1 CPU Frequency

To determine the influence of the CPU frequency, we record its course three times in our CPU parameter measurement. Figure 6.24 shows the CPU frequency curve of the DUT 1 processor during a CPU parameter measurement. Each point corresponds to the average CPU frequency in MHz at the respective data rate. We show the standard deviation with the help of the error bars. For this CPU parameter measurement setup, we did not configure any firewall rules on the DUT. We generate additional cross traffic by opening and closing SSH connections during each measurement run. The SSH connections allow us to start and stop the hardware parameter logging. The curve in Figure 6.24 shows that the frequency fluctuates between $1,592.5\ MHz$ and $1,595.5\ MHz$ but changes only minimally in absolute terms. At low data rates, the CPU frequency is not below the one for high data rates. Even under overload at 1000 MBit/s, which corresponds to the dark red area in Figure 6.3, the CPU frequency does not increase. This shows that the CPU frequency does not affect the latency result in a basic measurement.

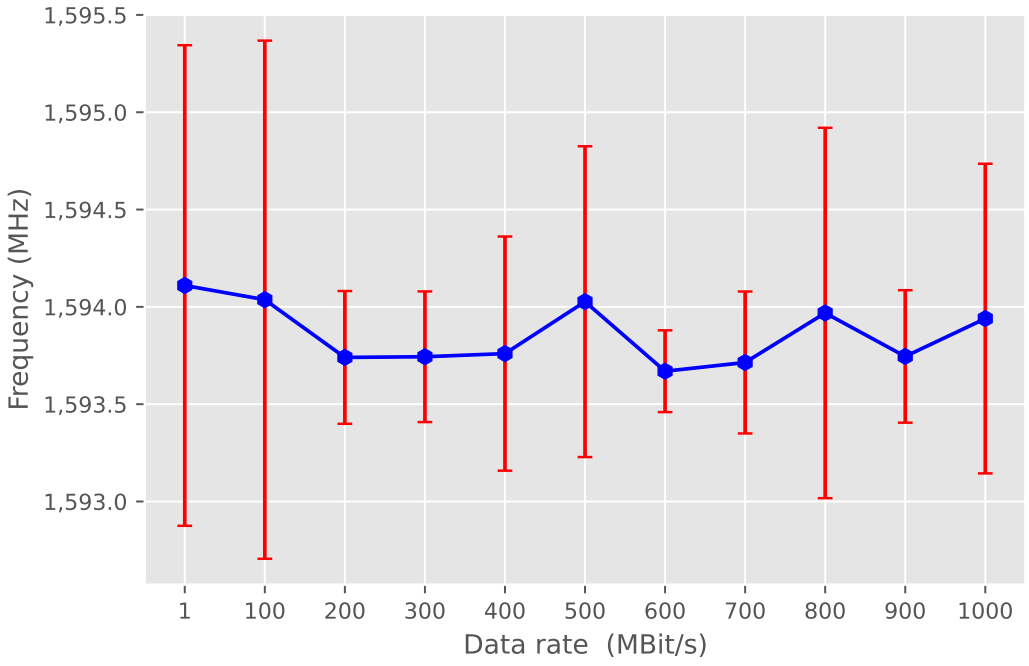


Figure 6.24: Average CPU Frequency in MHz during a base measurement with an average packet size of 160 Bytes on DUT 1

6.3.2 CPU Temperature

Since the temperature has an impact on the behavior of electronic devices, we investigate how it affects latency and packet loss on our DUT. We measure at 22 degrees Celsius ambient temperature without firewall rule configuration and only cross traffic through SSH connections. We show the CPU temperature curve during our CPU parameter measurement in Figure 6.25. Under normal ambient temperature conditions, the CPU temperature increases only less than 1 degree Celsius during our CPU parameter measurement. The CPU temperature, therefore, hardly increases during our CPU parameter measurement. In other words, the network traffic does not increase the CPU temperature. The increase in network traffic does not trigger a self-perpetuating process in which the CPU temperature continues to rise. We investigate the temperature influence on the performance in Section 6.4 in more detail.

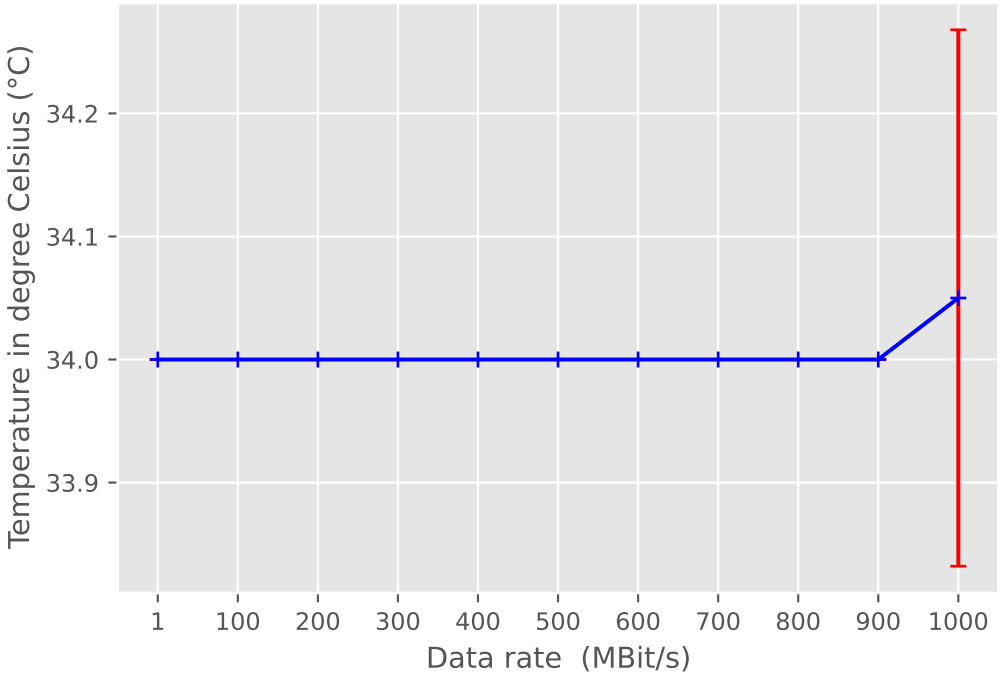


Figure 6.25: CPU Temperature in degree Celsius during a base measurement with an average packet size of 160 Bytes on DUT 1

6.3.3 Memory Usage

We measure the memory usage to trace possible packet loss back to excessive memory usage. Figure 6.26 displays the memory usage behavior during the same measurement we used for generating Figure 6.24 and Figure 6.25. Memory utilization is constantly increasing but is relatively low at a maximum of 5.9%. Despite the additional cross traffic, there are no outliers in the memory utilization. The overload range does, therefore, not lead to an overload of the memory.

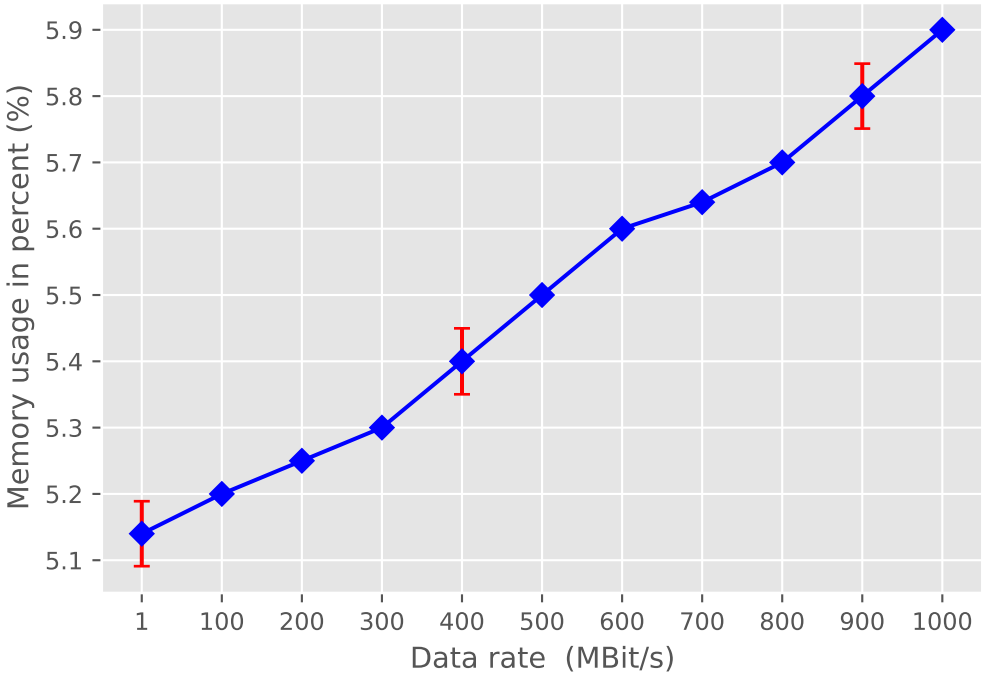


Figure 6.26: Memory usage in % during a base measurement with an average packet size of 160 Bytes on DUT 1

6.3.4 Packet Loss

In this section, we put our previous results from the CPU parameter measurement in relation to the packet loss. Figure 6.27 displays the corresponding packet loss for this CPU parameter measurement. If we compare the knowledge gained about the various hardware parameters with the packet loss, it becomes clear that the parameters have no direct influence on the packet loss. Packet loss occurs in our CPU measurement above 300 MBit/s and then continues to increase. In comparison, memory consumption increases linearly. The temperature hardly increases during a measurement. Despite increasing packet losses, the processor frequency barely changes. Thus, under normal conditions, the measured CPU parameters have no direct influence on the packet loss of the DUT. For our modeling, we do not need to consider the CPU frequency, CPU temperature, and memory usage.

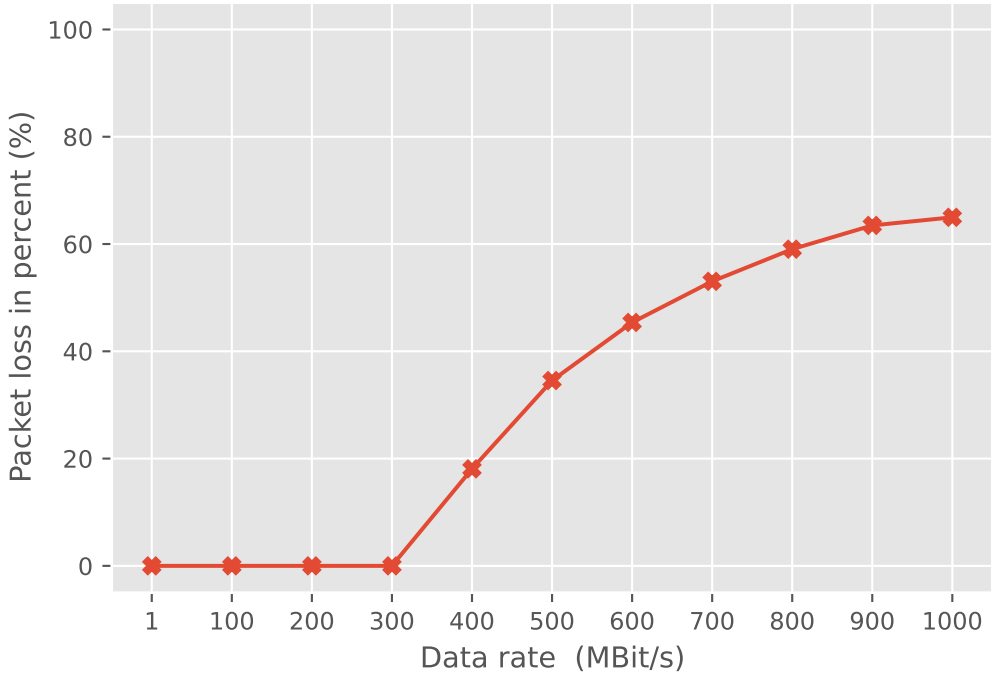


Figure 6.27: Packet loss in % during a base measurement on DUT 1

6.4 Temperature Influence

In this section, we describe in detail why we do not consider the temperature influence on the DUT for our modeling. To determine how far the DUT depends on the temperature, we focus on the vendor-specific temperature window for the DUT. Since the temperature window of the tested EAGLE40 goes up to 105 degrees Celsius [43], we heat it up to a temperature window of 97 to 120 degrees Celsius. This results in an accurate picture of the threshold range of the DUT.

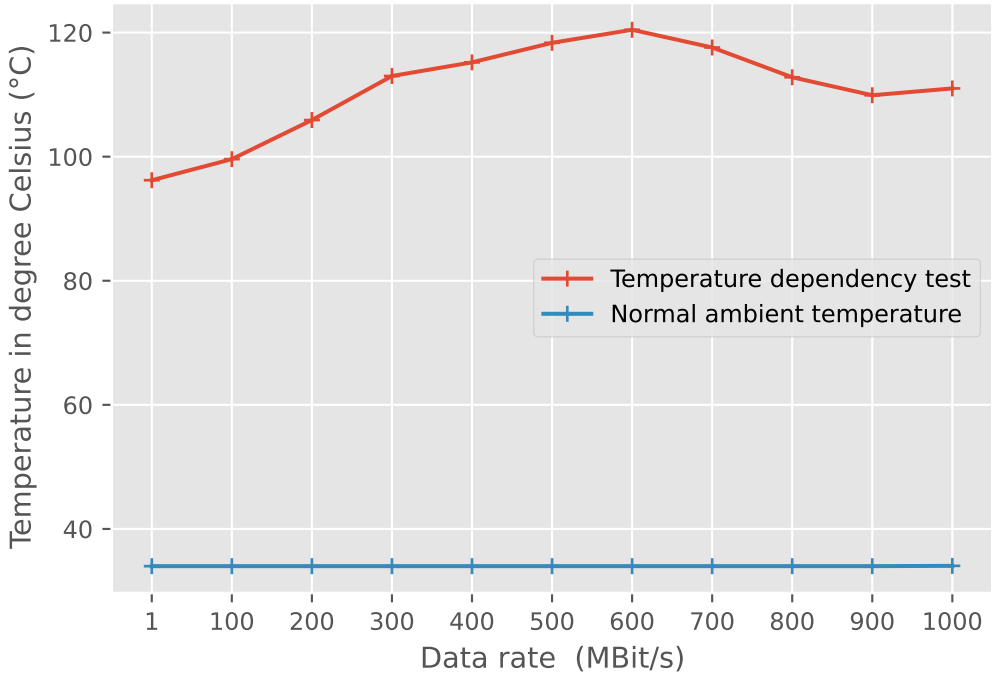


Figure 6.28: DUT 1 temperature curve comparison between temperature dependency test and normal ambient temperature (22 degrees Celsius)

Figure 6.28 shows two temperature curves, one for our temperature dependency test measurements and one for our measurement with normal ambient temperature. At normal ambient temperature (22 degrees Celsius), the CPU has a temperature range between 34 and 35 degrees Celsius. During our "Temperature dependency test" in Figure 6.28, we applied different temperatures for each data rate. At 200 MBit/s, we reach the specified limit of 105 degrees Celsius. To investigate this threshold in more detail,Figure 6.29 shows the observed measurement points at 200 MBit/s. The maximum temperature we generate on the DUT is 120 degrees Celsius at 600 MBit/s.

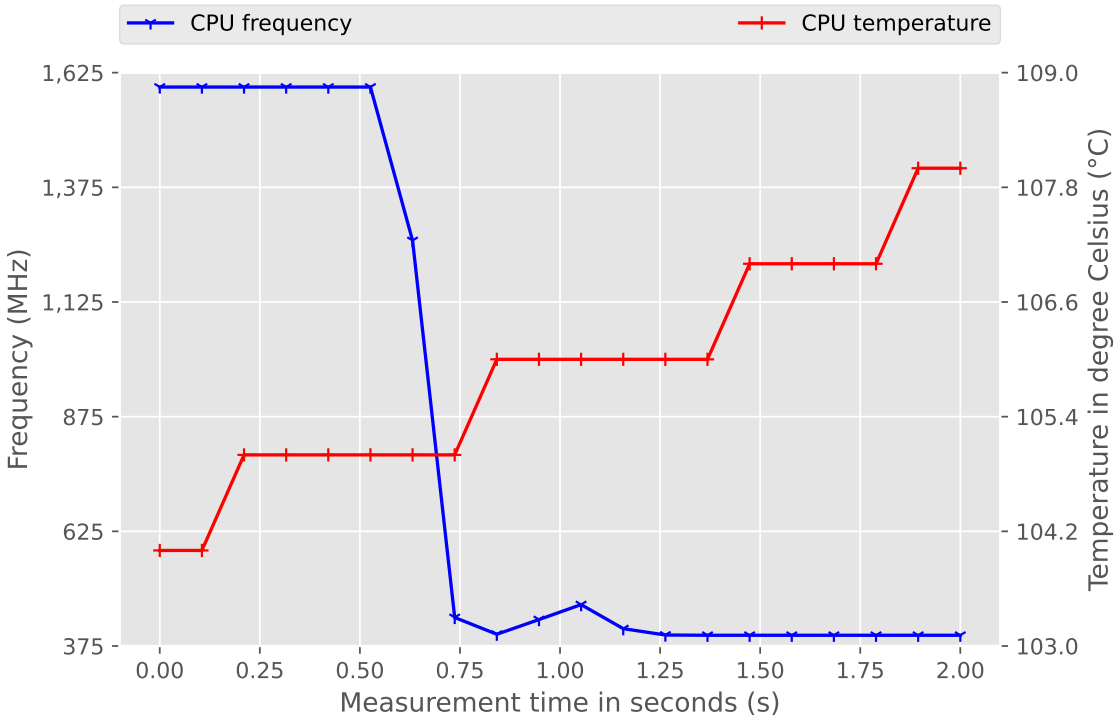


Figure 6.29: DUT 1 measurement with 200 MBit/s data rate and 64 Bytes packet size; Record time 2 seconds; Comparison between temperature increase and CPU frequency decrease

Figure 6.29 shows that the CPU frequency and the temperature curve in comparison. The CPU frequency remains constant at around $1,600\ MHz$ below 105 degrees Celsius CPU temperature. As soon as the temperature rises further, the CPU frequency drops drastically. The firewall tries to prevent damage from overheating by lowering the frequency to $\approx 400\ MHz$. This changes the CPU frequency behavior of the DUT. During our measurements under normal ambient temperature, the CPU frequency barely changes, as we describe in Section 6.3.2.

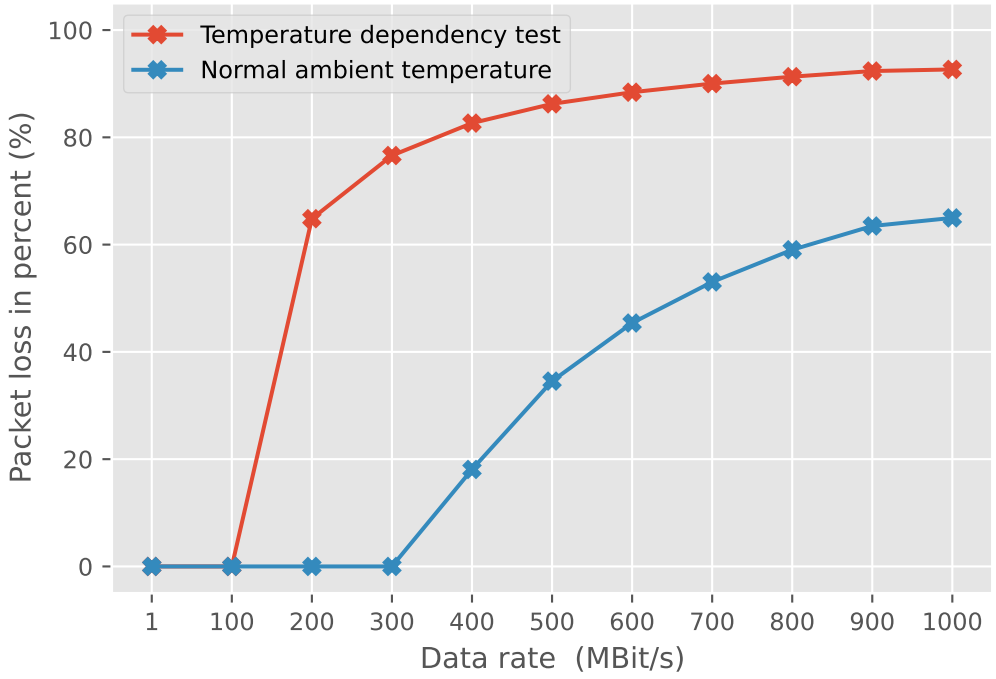


Figure 6.30: DUT 1 packet loss comparison between normal ambient temperature and temperature dependency test

In parallel to the decreasing CPU frequency, the packet loss increases. We show this in Figure 6.30. As soon as the temperature exceeds 105 degrees Celsius, the packet loss increases rapidly to over 80 %. The „Normal ambient temperature“ in Figure 6.30 describes the CPUs packet loss during a measurement with 22 degrees Celsius ambient temperature. The packet loss starts at 300 MBit/s and increases to around 65 %. We present the CPU temperature behavior at 22 degrees Celsius ambient temperature in Figure 6.25. Compared to the normal ambient temperature, the packet loss in our temperature dependency test increases earlier and more steeply.

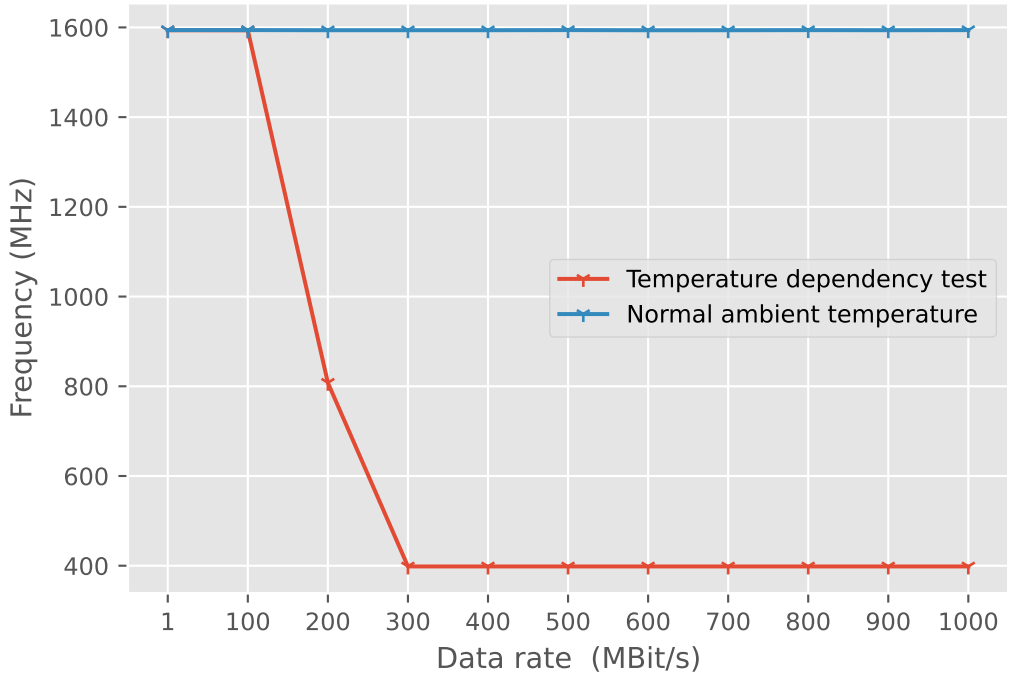


Figure 6.31: DUT 1 frequency comparison between normal temperature and temperature dependency test

To exclude a general drop in frequency at higher data rates, we compare the CPU frequencies between our normal ambient temperature measurement and our temperature dependency test measurement in Figure 6.31. This shows that the CPU's frequency barely changes under normal ambient temperature conditions. The frequency remains at 1600 MHz. The high temperature forces the CPU to reduce the frequency. When we increase the temperature, the frequency drops to 400 MHz. The extreme temperature causes packet loss on the DUT.

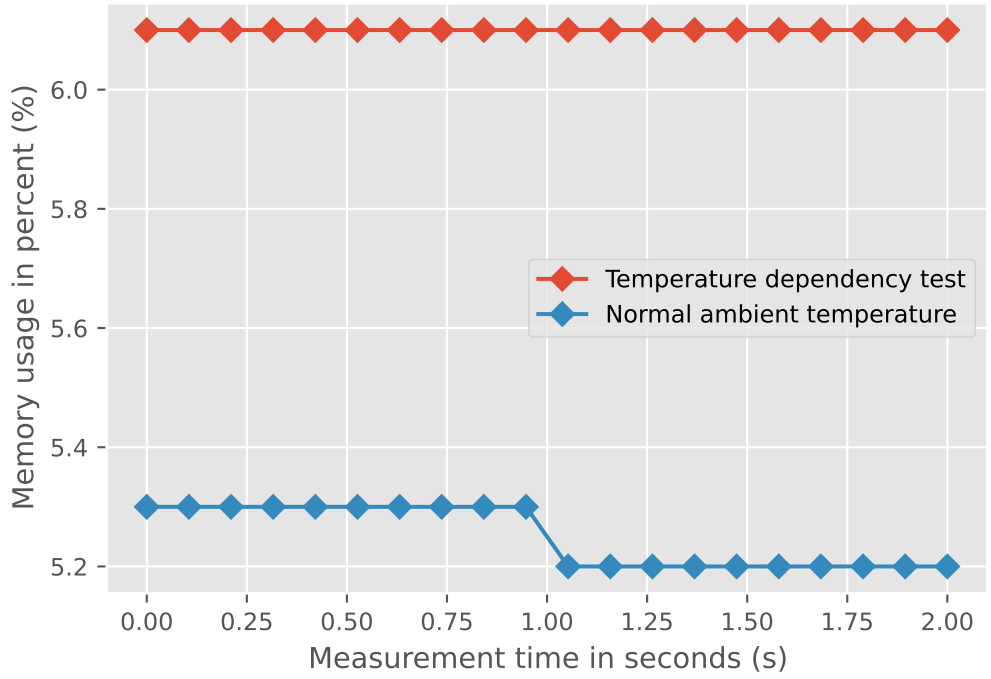


Figure 6.32: DUT 1 memory usage in % comparison between normal temperature and temperature dependency test

To learn more about the effects of temperature, we also examine the memory usage behavior of the DUT. For this purpose, we compare the memory consumption with increased CPU temperature and without increased temperature in Figure 6.32. The figure shows the memory consumption during the temperature threshold measurement at 200 MBit/s. It is noticeable that there is no increase in memory utilization despite the increase in temperature. The memory utilization does not change during the temperature dependency test as well as during the normal ambient temperature test. The high temperature has no negative effect on memory. This means that the packet loss and thus the increase in latency is directly related to the reduction in CPU frequency.

We thus show in Figure 6.29, Figure 6.30 and Figure 6.31 that only temperatures above 105 degrees Celsius have a direct influence on the latency. As our EAGLE 40 is only approved for temperatures up to 105 degrees Celsius, these results have an impact on our model definition. As for each DUT, the allowed temperature can vary, we define our model only for the respective operating temperature window. Temperatures above the operating temperature window are not part of the normal working environment. Therefore, we do not consider these extreme cases in our model. In our model, we assume that the ambient temperature is within the permitted operating window.

6.5 Prediction Model

In this section, we explain in detail how we build our latency prediction model. We start by describing how we select our prediction function. Second, we describe the latency behavior prediction. Third, we extend our model with the ability to predict firewall configuration effects. Fourth, we show how our model handles additional information about network traffic. Afterward, we discuss why it is necessary that our model can predict the latency in a time interval. The next step in our model is the ability to predict packet loss. We then show how to automate the model finding process and test its robustness. Finally, we address the accuracy of our model.

6.5.1 Selection of the Prediction Function

In order to build a model for firewalls, we need some basic measurement data about the DUTs. We generate the basic measurement data through base measurements. Based on these measurement results, we determine the appropriate prediction function.

The following procedure helps us to find a suitable prediction function:

1. Evaluation of multiple measurement results
2. Determination of a suitable measurement
3. Determination of a selection of prediction functions that fit the measurement data
4. Calculation of the function parameters
5. Optical comparison and calculation of the square distance
6. Adjustment test
7. Selection of the prediction function that performs best at 5. and 6.

To obtain reliable data, we repeat the measurements several times to eliminate possible measurement errors. We combine the latency and packet loss measurement results in a reliable measurement image that serves as a basis for the selection of a prediction function. Once we determine a set of suitable prediction functions, we calculate their function parameters. Next, we compare the fitted prediction function visually with the original measurement data. To determine the accuracy, we calculate the squared distance. We change the function parameters to increase the accuracy of the prediction function. Through testing different prediction functions with different parameters, we find the prediction function that fits the measurement results best. We use the best prediction function for our further modeling.

6.5.2 Normal Latency Prediction

In this section, we model the basic DUT behavior with our traffic. In Section 6.5.1, we describe the procedure to obtain a suitable function. This also includes a visual

comparison of the measurement results. At the beginning of the implementation section, we consider the factors packet size and data rate individually. To graphically represent the behavior of the DUT, we create a 3D model. The resulting 3D model in Figure 6.3 shows the influence of the two parameters on the latency.

Figure 6.3 shows the latency curve across the data rate and the packet size. We show how the combination of a high data rate and a small packet size leads to an extreme increase in latency. Combining the two parameters reduces the complexity of our model. To do so, we combine both parameters in the common unit pps. As a result, we have a more straightforward presentation of the measurement results. This makes modeling easier because we are looking at one parameter instead of two.

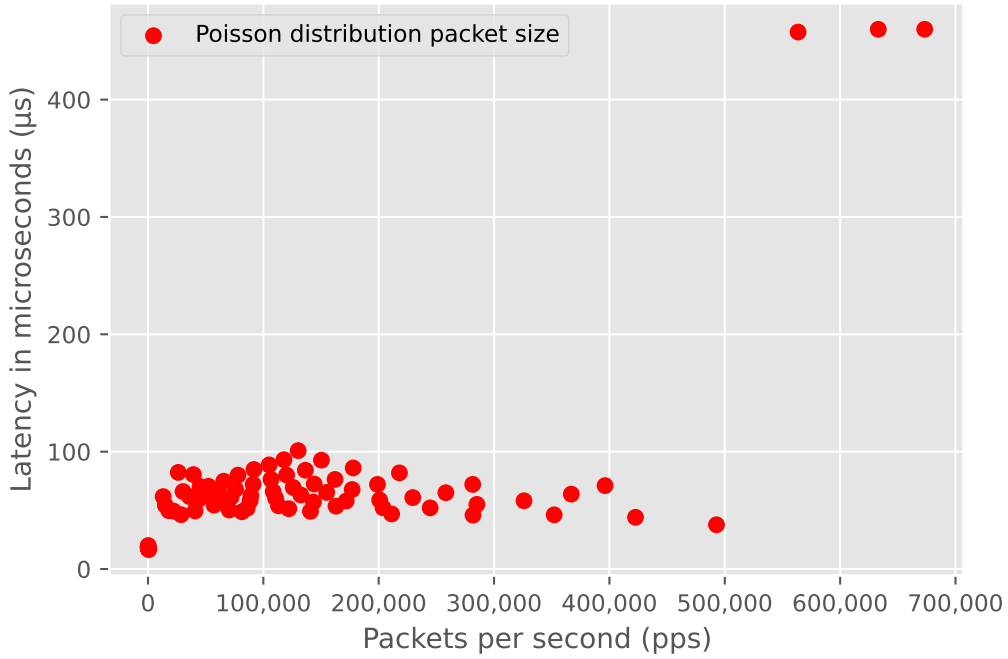


Figure 6.33: DUT 1 base latency measurement in μs with no firewall rules configured

In Figure 6.33, we show how the simplified representation with pps looks like. We created those with a basic measurement on the EAGLE40 using iptables (DUT 1). During a basic measurement, we configure no firewall rules on the DUT. Each point in the figure represents the mean latency of the DUT for a corresponding pps configuration. The lowest latencies are 16.6 μs (below $1,000$ pps) and the highest are 459.95 μs (above $620,000$ pps).

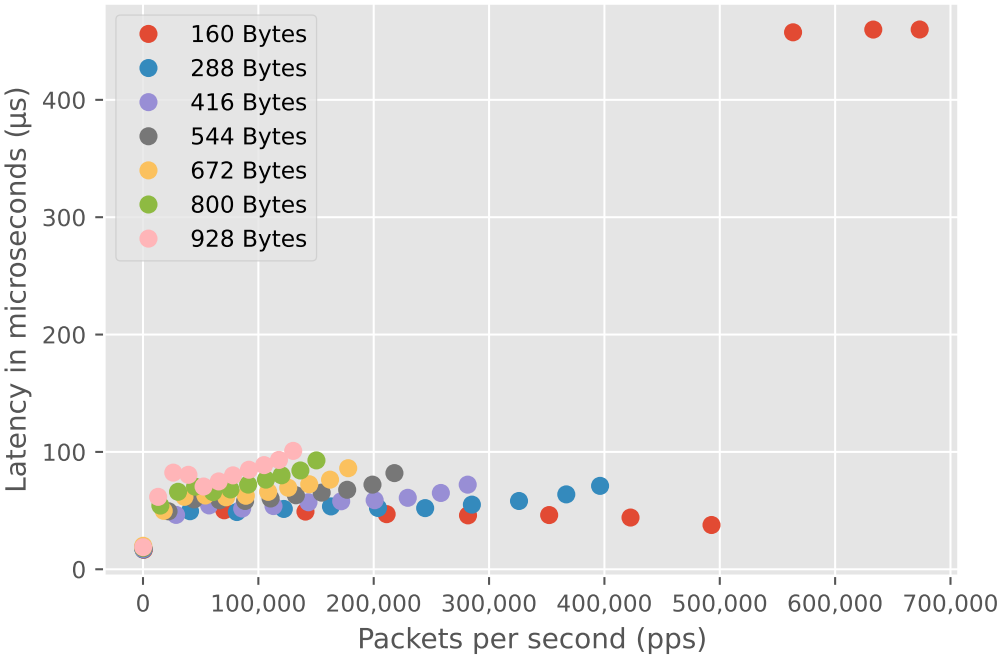


Figure 6.34: DUT 1 base latency measurement in μs average packet size distribution with no firewall rules configured

The measurement results show that the latency of the iptables firewall on the DUT increases suddenly. From this point on, the latency remains constantly high. We refer to this area as overload area. In Figure 6.34, we show the same measurement again. We distinguish the measurement results in Figure 6.34 by their average packet size. We notice that the overload area only occurs with the small packet size range. Because there are too many packets that have to be processed by the firewall. Knowing why the firewall enters the overload area is important for our model.

Our model is intended to predict the behavior of a firewall. To do this, we use functions that help us model the latency behavior. To find the ideal function, several iterations and adjustments are necessary until the result is satisfying.

At the beginning of our research, we considered modeling with GPR. The ability to enter an unlimited number of parameters into the regression seemed promising. This way, we can use all measurable parameters of the firewall for the modeling without measuring their relevance. Unfortunately, the GPR requires a large amount of training data to provide adequate results. For us, this would mean that profiling a firewall becomes much more complex. The increasing complexity leads to long measurement times. To create an accurate model, our profiling has to test almost every possible configuration of the firewall. Such a scenario is not realistic. It is not part of our model to test all firewall configurations in advance. Since the GPR chooses the function itself, it can happen that the GPR chooses a polynomial function with a high degree. Unfavorable positioning of two measurement points can cause the phenomenon of Runge as described in Section 3.6. This results in incorrect predictions. Due to these disadvantages, we do not pursue modeling with

the help of the GPR.

Modeling with the help of self-selected functions, on the other hand, is more promising. With the help of curve fit, we can fit various functions to given data points. The SciPy curve fit function calculates the function parameters for the function we select. Before we calculate the parameters, we need to choose a function that represents the behavior best. Since the measurement points cover two latency ranges, a sigmoid function seems like a good choice. To illustrate this graphically, we show in Figure 6.35 what the adjusted sigmoid function looks like. The sigmoid function is constant from 0 pps to about 550,000 pps, then the function increases linearly from about 18.3 μs to 454.7 μs. From $\approx 580,000$ pps, the function remains constant again. With a few measuring points, we can represent the characteristics of the DUT in a function. However, the sigmoid function has several disadvantages for our modeling. First, the overload increase in latency for the tested firewall is not continuous, but abrupt. When using the sigmoid function in our modeling, we cannot represent a sudden latency increase. The prediction of latencies in the transition region leads to significant deviations from reality. Another disadvantage of the sigmoid function is the lack of possibility to model the non-overload area more precisely.

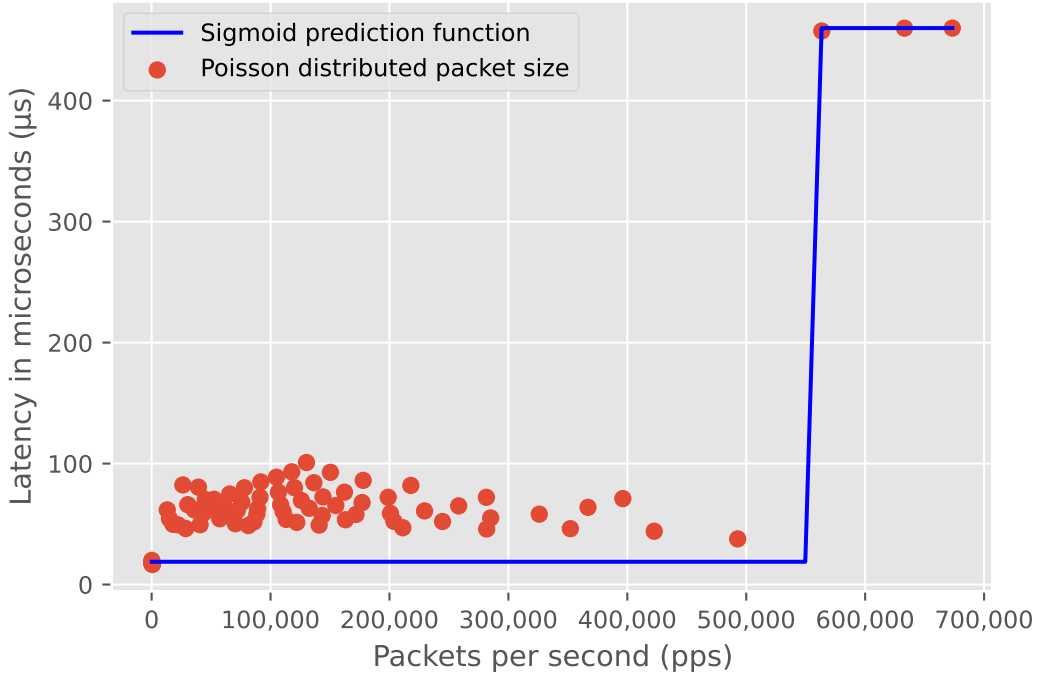


Figure 6.35: DUT 1 base measurement with sigmoid prediction function

Our measurement results in Figure 6.35 show a large variance between the function and the measurement results, especially in the range of fewer than 300,000 pps. We can move the lower range of the sigmoid function upwards or downwards on the

y axis. However, the deviations are still large. Due to the properties of the sigmoid function, we can't model the non-overload are more flexibly.

To better model not only the overall behavior but also the individual load areas, it makes sense to cluster the data points. For this modeling approach, we use two prediction functions in Figure 6.36. During the creation of this model, we divide the data points into an overload and a non-overload area. The blue line represents the non-overload area prediction function, and the green line represents the overload area prediction function. We represent the non-overload and overload area in this example by linear functions. Each function represents the respective area as well as possible.

The transition to the firewalls overload area is directly visible. The separation enables us to map the sudden increase in latency without a transition period. In this type of modeling, we notice that the variance is large, especially in the non-overload area. Accordingly, we continue testing various other functions for this area to find a better representation.

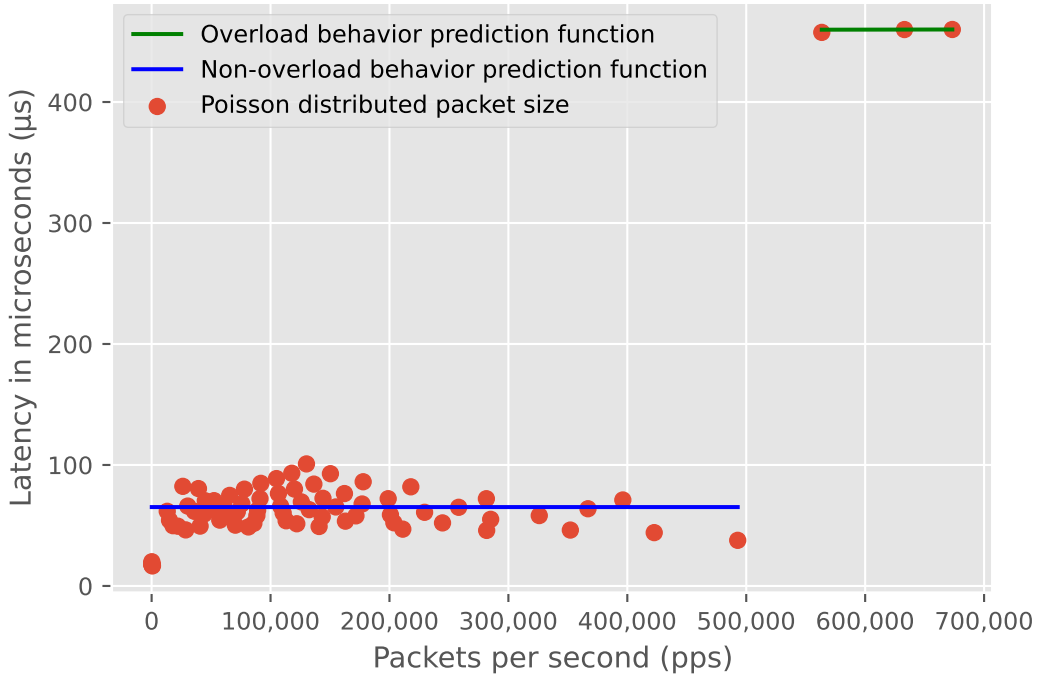


Figure 6.36: DUT 1 base measurement with two linear prediction function

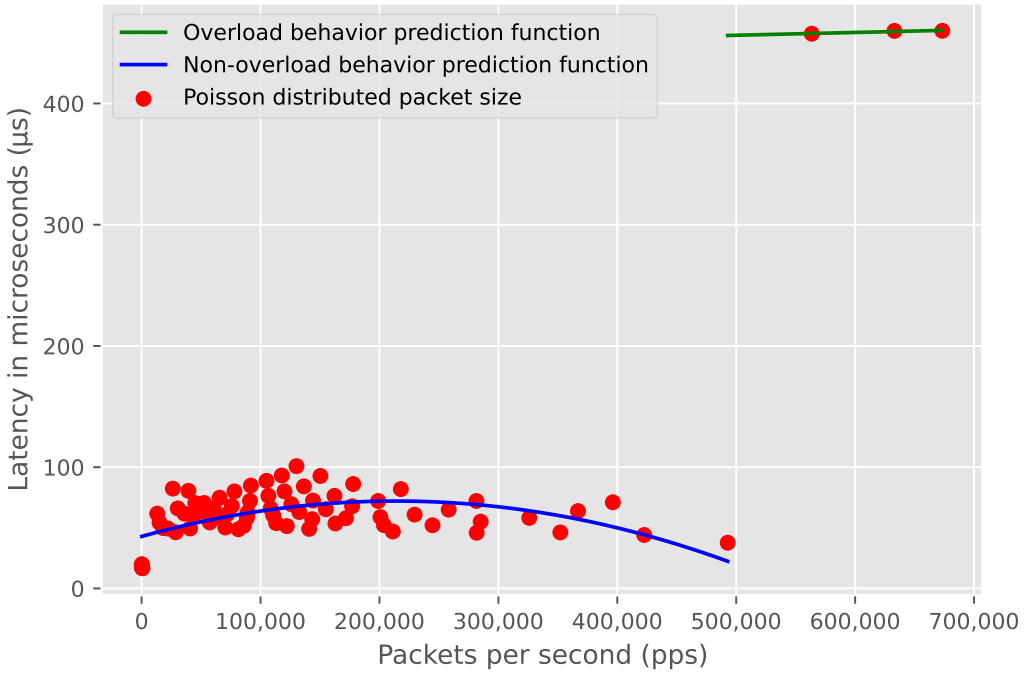


Figure 6.37: DUT 1 base measurement with a quadratic and linear prediction function

Curve fit with a quadratic function allows a better mapping of the latency behavior of the DUT. We show a model with a quadratic function (blue line) for the non-overload area prediction in Figure 6.37. In addition, we extend the function of the overload area, so we can predict more conservatively in the threshold area. Without this adjustment, our model predicts a much lower latency in the overload threshold area. An under-predicted latency is much worse than an over-predicted latency. In the worst case, an application expects a packet but due to a too aggressive latency prediction, the packet does not arrive at the application. We, therefore, tend to predict higher latencies.

We use the gained knowledge to define our functions for latency prediction. We formally present our modeling functions in Equation (6.7). The variable x determines the pps value for which we predict the latency. For all pps points up to the overload threshold we use a quadratic function for the latency prediction. The function $g(n)$ (Equation (6.2)) determines this overload threshold. Above the overload threshold, we use a linear function to describe the overload area. With the variable n we specify the average matching position of the firewall rules.

$$f(x) = \begin{cases} ax + b & x \geq g(n) \\ cx^2 + dx + e & x < g(n) \end{cases} \tag{6.7}$$

$$f(x) \in \mathbb{R}^+ \tag{6.8}$$

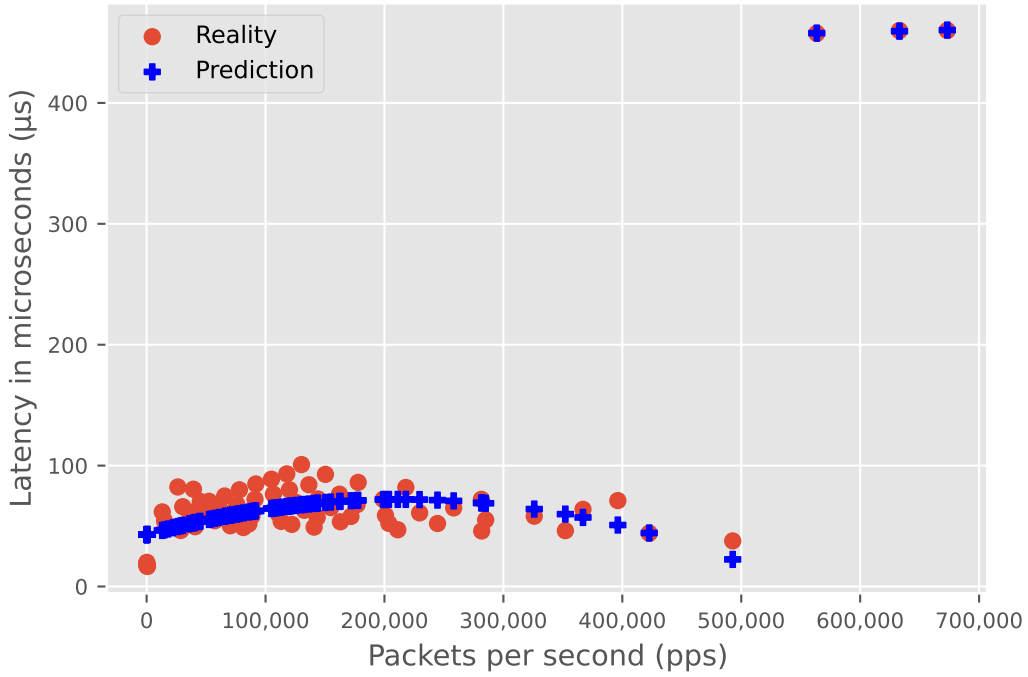


Figure 6.38: Predicted latency for our DUT 1 base measurement without firewall rules; The red points originate from our base measurement without firewall rules

The quadratic function represents the non-overload area very well. To show this, we plot in Figure 6.38 the latencies we predict for the base latency measurement. Each red point in the figure corresponds to the real latency that we measured, and the pluses correspond to the latency prediction by our model. In the non-overload area, our predictions have a larger spread from the real latencies than in the overload area.

To better illustrate the deviation between our prediction and the real measured latency, we use the letter-value plot in Figure 6.39. We explain the detailed structure of a letter-value plot in Section 3.5. We observe that most deviations are close to 0 μs. The letter-value plot shows that most of our predictions have a minimal deviation. However, some predictions have a deviation of almost 35 μs. The mean of all deviations for this measurement is ≈ 0 μs. With a probability error of 5 %, the true mean is between -3.43 μs and 3.43 μs. For smaller packet sizes, the mean value of our deviation is -7.77 μs. This means we predict for small packets a higher latency compared to the real latency. We explain the improvements we made to our model with additional network traffic information in Section 6.5.4.

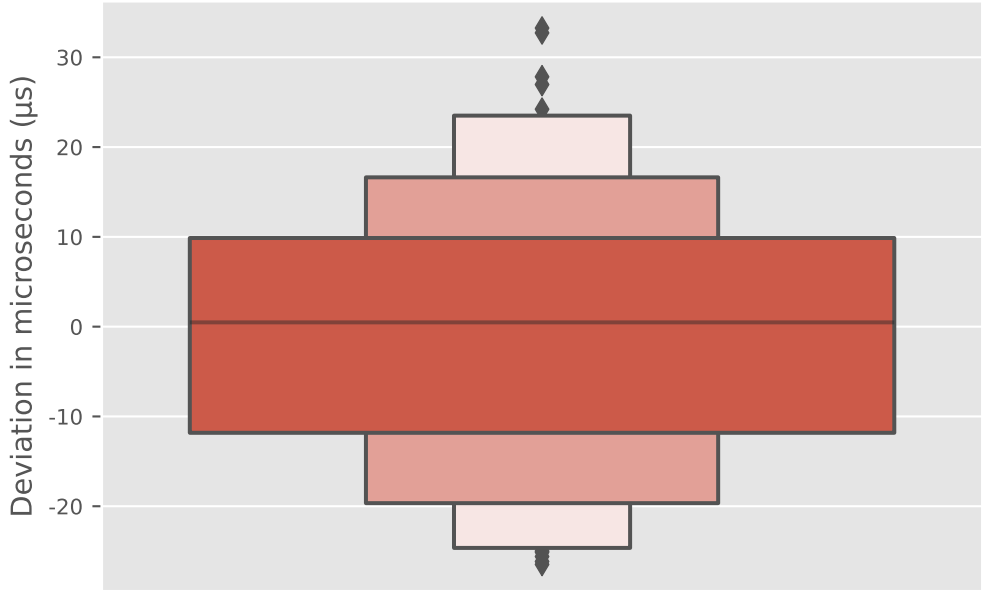


Figure 6.39: Deviation between our latency prediction and the latency we observed through our profiling on the DUT 1

6.5.3 Firewall Rule Dependent Latency Prediction

So far, our modeling can only predict the latencies without the influence of the firewall rule. To properly incorporate the influence into our model, we need to consider the knowledge from Section 6.2.1. Firewall rules shift the overload threshold and overall latency of a DUT. In Equation (6.2) we already demonstrated how we use the firewall rule matching position to predict the overload threshold. In Section 6.5.2 we use the threshold to determine the prediction function. Since we know from Section 6.2.1 that the matching position n also influences the latency, we have to adjust our function parameters even further. Using the firewall rule matching position, we determine the firewall rule-dependent latency offset. We show our adjusted functions in Equation (6.9).

$$f(x) = \begin{cases} ax + h(n) & x \geq g(n) \\ cx^2 + dx + h(n) & x < g(n) \end{cases} \tag{6.9}$$

Function $h(n)$ indicates the latency offset. We use this offset in our functions to shift the latency prediction in the y direction. This way we map the firewall rule-dependent factor in our model.

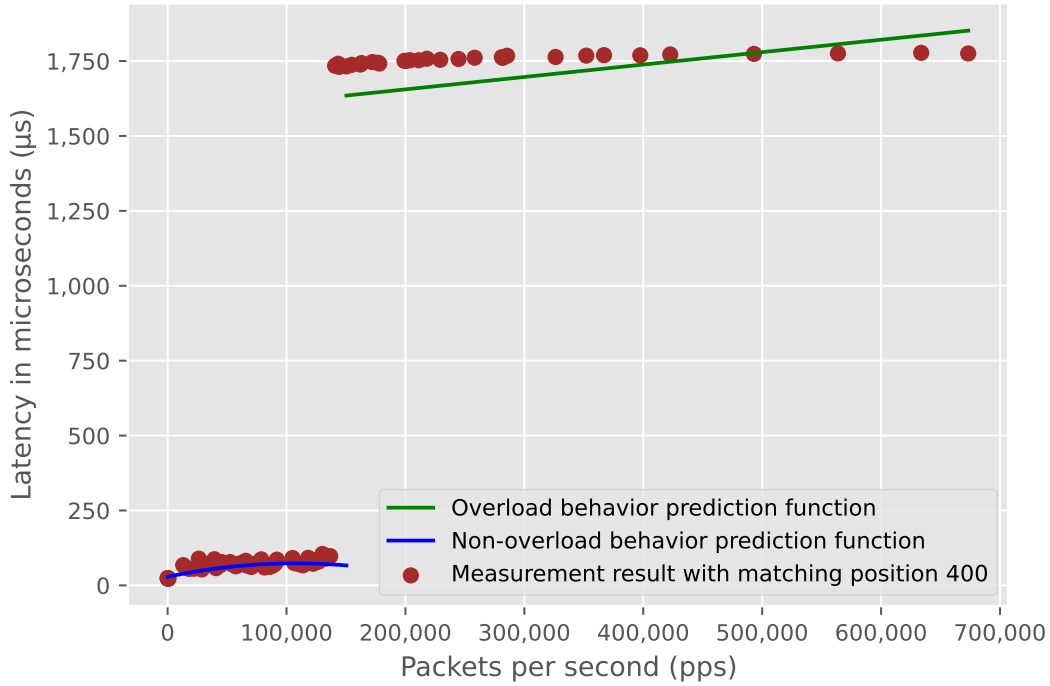


Figure 6.40: Firewall rule dependent prediction on DUT 1 with firewall rule matching position at 400; The brown points originate from our profiling with 400 firewall rules

The functions are shifted appropriately in the y and x direction by our adjustments. We show an example of our prediction with firewall rule configuration in Figure 6.40. The brown dots represent the mean latencies we measure with 400 firewall rules. Our model predicts the DUT 1's behavior with 400 firewall rules configured. The 400th rule is the matching rule. In comparison to our prediction, we show what the behavior looks like in reality. We generated the data points on the DUT 1 by measuring with 400 firewall rules configured. We show that our model can reproduce the behavior of the DUT 1. Through our model adaptations, we also enable the user to predict the behavior of the DUT with firewall rules.

6.5.4 Latency Prediction with Additional Network Knowledge

So far, our model can predict the behavior of the DUT based on general network information. We do not take into account whether we generate mainly certain packet groups in the network. Figure 6.34 shows that the latencies we measure are divided into different packet sizes. If we know the packet sizes, our model deviates significantly from the actual latency, especially in the non-overload area. We introduce an optimization in our model where a prediction is made in the non-overload area for small and large packet sizes. A more accurate prediction of the

non-overload area helps us to meet the requirements of industrial networks and real-time traffic.

In Figure 6.41, the orange points represent the latencies of the large packets and the violet points represent the latencies of the small packet sizes. Under non-overload conditions, large packets have a higher latency than small packets. This is the result of the physical transmission time of larger packets. To ensure that the additional knowledge about network traffic does not remain unused, we offer optimization for our model. We calculate two additional functions for the non-overload area. One function for rather small packets and the other function for rather large packets.

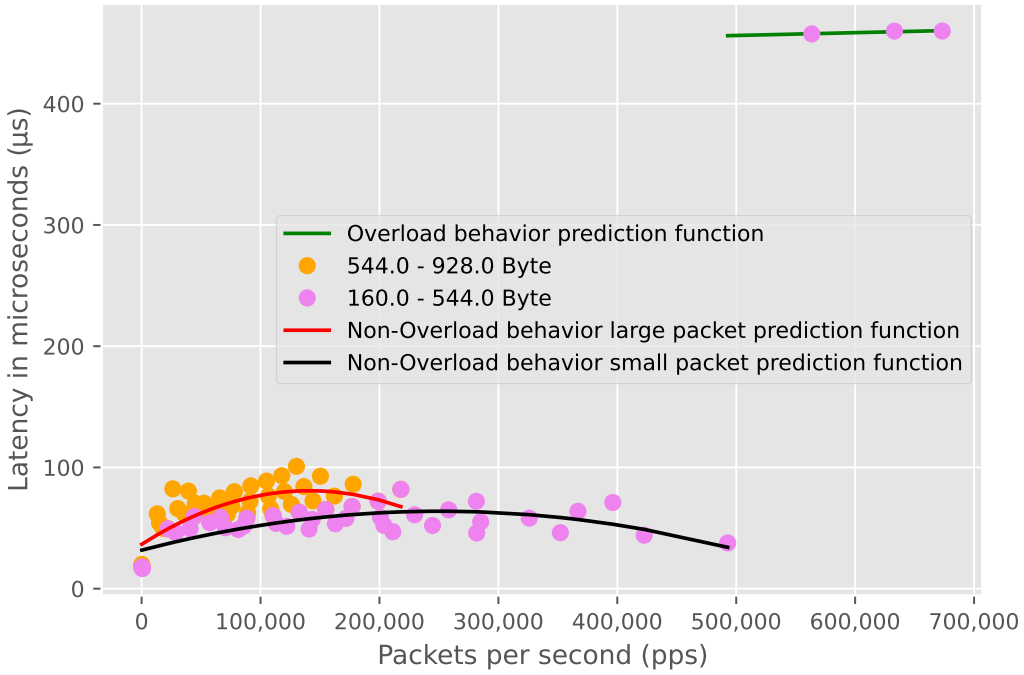


Figure 6.41: DUT 1 behavior prediction with additional network knowledge; The orange and violet points originate from our profiling without firewall rules

To avoid creating a separate function for each packet size, we divide the existing data into two groups. We group the smaller packets into 160-544 Bytes. Furthermore, we group the large packets as a 544-928 Bytes group. The two groups overlap intentionally. The overlapping packet sizes form the medium-sized packets. In addition, we do not have to choose a fixed limit but rather allocate the packets to the groups on a percentage basis. We take half of the measured packet sizes as the threshold value. We multiply this value by 1.25 to get the upper limit of the packet sizes that come into the small packet size group. We collect all packet sizes larger than the threshold in the large packet size group. In our example, the 544 Bytes packets form the intersection between the two groups.

With the additional information about the network traffic, our model can predict the behavior of the DUT better. The two functions in the non-overload area are much more accurate than the normal prediction in Figure 6.37.

$$f_{small}(x) = ax^2 + bx + h_{small}(n) \tag{6.10}$$

$$f_{large}(x) = ax^2 + bx + h_{large}(n) \tag{6.11}$$

$$f_{large}(x) \in \mathbb{R}^+ \tag{6.12}$$

$$f_{small}(x) \in \mathbb{R}^+ \tag{6.13}$$

$$x \in \mathbb{Z}^+ \tag{6.14}$$

$$\tag{6.15}$$

In Equation (6.10) and Equation (6.11), we show the functions for our non-overload area prediction with additional network knowledge. Equation (6.10) displays the latency prediction of the small packet size group. The Equation (6.10) is the same we use in Equation (6.7). The difference is that we calculate the latency offset (Equation (6.5)), this time, based on the small packet sizes. We characterize this by the function $h_{small}(x)$. For Equation (6.11), we determine the latency offset (Equation (6.5)) using the large packet sizes, $h_{large}(x)$. Without constraints, the function could predict 0 μs, depending on the parameters selected. A latency of 0 μs is not realistic, hence our adjustment. If the user has additional information about the expected network traffic, we can use this information in our model. This makes our prediction more accurate for the specific network traffic.

6.5.5 Latency Prediction in a Time Interval

So far, our model can determine the static behavior at a time t. We do not yet consider changes in the traffic model, besides the packet size, during a measurement run. For our model, we not only predict the higher latencies after the data rate increase but also model the spike.

In Figure 6.42 we show how the prediction of the mean latency spikes looks. Each line represents a different data rate. We assume a prediction period of 43 seconds. The Poisson distribution range for our packet sizes is constant through this prediction period. We start with a data rate of 20 MBit/s. After the first 4 seconds, we assume that we increase the latency by 100 MBit/s. We observe that our model predicts the spike. Then we assume that the data rate remains constant. After 5 seconds, we repeat the latency increase. We repeat this until we reach 820 MBit/s. Since our model stores the network history, it notices the data rate increase. Accordingly, the model adjusts its latency prediction. For each change in data rate from time $t-1$ to time t, our model predicts a latency spike. That is why we show several latency spikes in Figure 6.42.

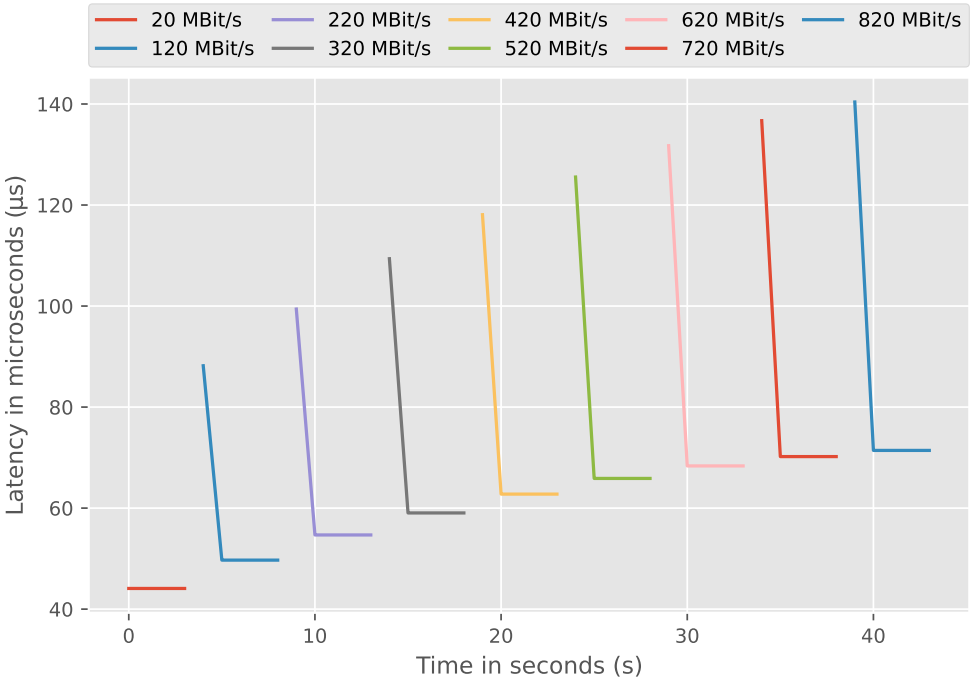


Figure 6.42: Predicted mean latency curve for the DUT 1 with data rate change; Prediction time interval 43 seconds

6.5.6 Packet Loss Prediction

Until now, our model can predict the latency of the DUT. Especially in industrial networks, latency alone is not an important feature. Industrial networks also need information about the packet loss behavior of the DUT. Therefore, a packet loss prediction is still missing in our model. To include the packet loss in our model, we first need to look at the packet loss in our previous latency measurements.

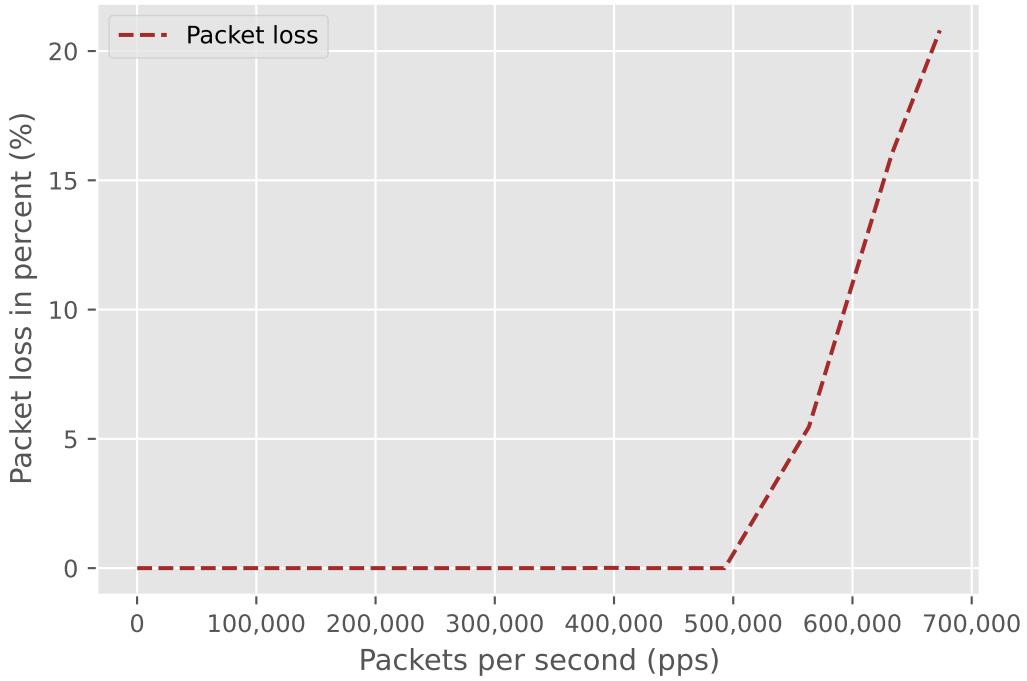


Figure 6.43: Packet loss during a basic measurement on the DUT 1

In Figure 6.43, we show the packet loss that we measured as a dashed line. We observe that the packet loss is mostly non-existent until it increases significantly at about 500,000 pps. By comparing the increase in packet loss with the latency measurement results in Figure 6.33, we see that the packet loss occurs when the firewall enters its overload area. To better understand packet loss, we examine what happens to packet loss after changing the firewall matching position. For this purpose, we investigate the packet loss behavior with different firewall rule configurations. We use the same firewall rule configurations as in Section 6.2.1.

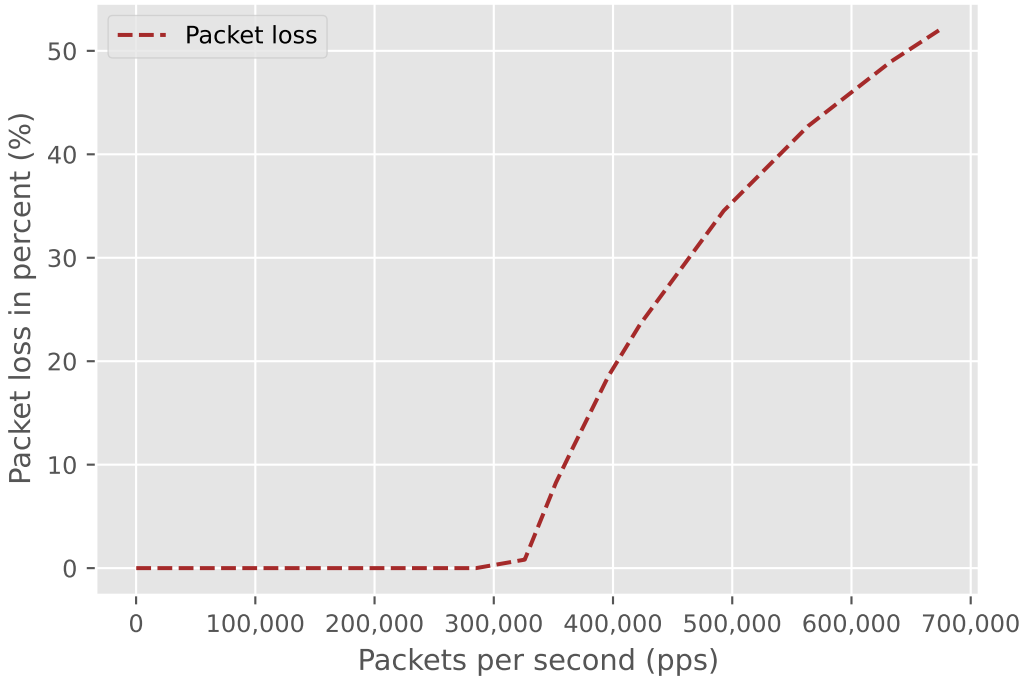


Figure 6.44: Packet loss during a measurement on the DUT 1 with average firewall rule matching position of 100

In Figure 6.44 we plot the packet loss of the DUT 1 at an average firewall rule matching position of 100. We can see directly that the packet loss starts to rise earlier and rises higher. From our packet loss measurement, we conclude that the packet loss starts to increase with the overload threshold. Accordingly, we use Equation (6.2) to determine the start of packet loss.

In order to be able to predict the course of packet loss, it is not only important to know the start of packet loss, but also how high the packet loss will rise. To define the packet loss depending on the fire wall rule matching position, we need to determine the packet loss behavior across multiple firewall rule configurations. Figure 6.45 shows the maximum measured packet loss from the firewall rule measurements. Every x represents the maximum packet loss which we measure with the corresponding firewall rule configuration. We measure maximum packet losses from 20.83 % to 89.6 %. We fit a root function to the packet losses that we observe. The dashed line in Figure 6.45 is our adjusted root function. The adjusted root function represents the maximum packet loss behavior. Using this root function, we can determine the maximum packet loss for the configured firewall rule matching position. With this knowledge we can then model the packet loss behavior.

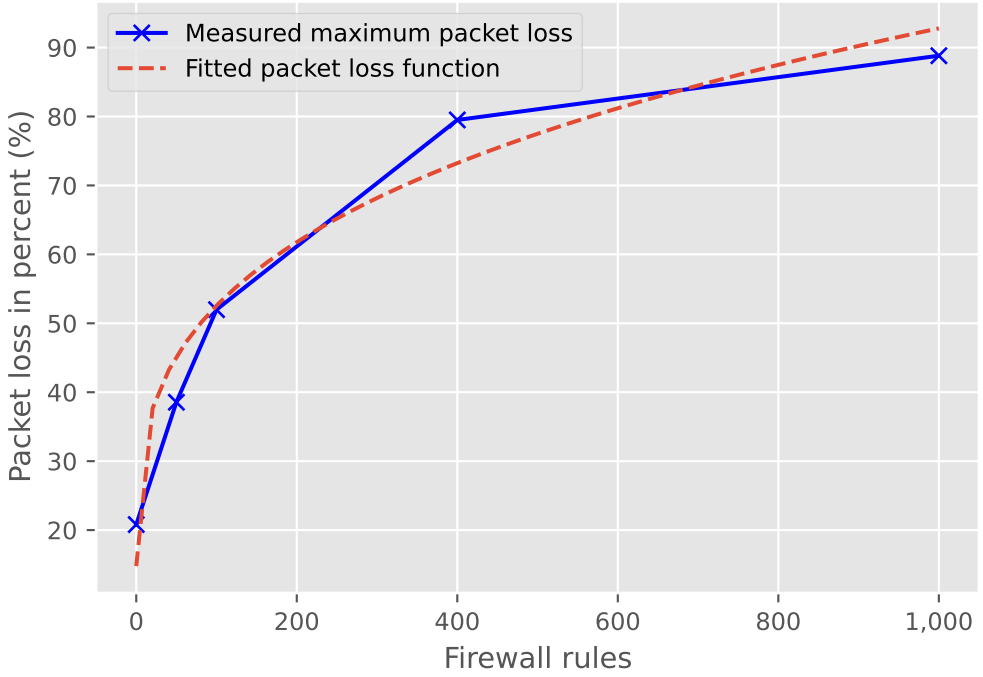


Figure 6.45: Maximum packet loss prediction function for the DUT 1

We define our root function in Equation (6.16). We limit the range of values from 0 to less than 1. Because packet loss cannot be less than 0. Furthermore, packet loss of 1 means that we cannot establish a connection.

$$m(n) = b \cdot x^a + c \tag{6.16}$$

$$\{m(n) \mid 0 \leq m(n) < 1\} \tag{6.17}$$

From the threshold and the maximum packet loss, we determine a quadratic function that predicts the packet loss. The maximum possible packet loss forms the apex of the function. We show our function for predicting packet loss in Equation (6.18). For the low pps range up to the overload threshold $g(n)$ we calculate a packet loss of 0%. Above the overload threshold, we determine the packet loss using our quadratic function, which we fit using the maximum packet loss point and the threshold point. Our quadratic function is only defined in the value range from 0 to less than 1. The packet loss can never be less than 0 and a packet loss of 1 would be equivalent to no connection.

$$p(x) = \begin{cases} cx^2 + dx + e & x \geq g(n) \\ 0 & x < g(n) \end{cases} \tag{6.18}$$

$$\{p(x) \mid 0 \leq p(x) < 1\} \tag{6.19}$$

With the knowledge about the behavior of packet loss, we adjust our prediction model accordingly. In Figure 6.46, we place our packet loss prediction in the context of the packet latency measurement on the DUT 1. We show the results from our latency measurement on the DUT 1 without firewall rules. The dashed curve in Figure 6.46 is our packet loss prediction. We can observe that our model predicts packet loss greater than zero only above the overload threshold. The overload area begins with the packet loss.

We present the real packet loss that we observe in our latency measurement in Figure 6.43. Our model predicts a similar curve of packet loss as it occurs in reality. With these adjustments, our model is now also able to predict the packet loss of a DUT.

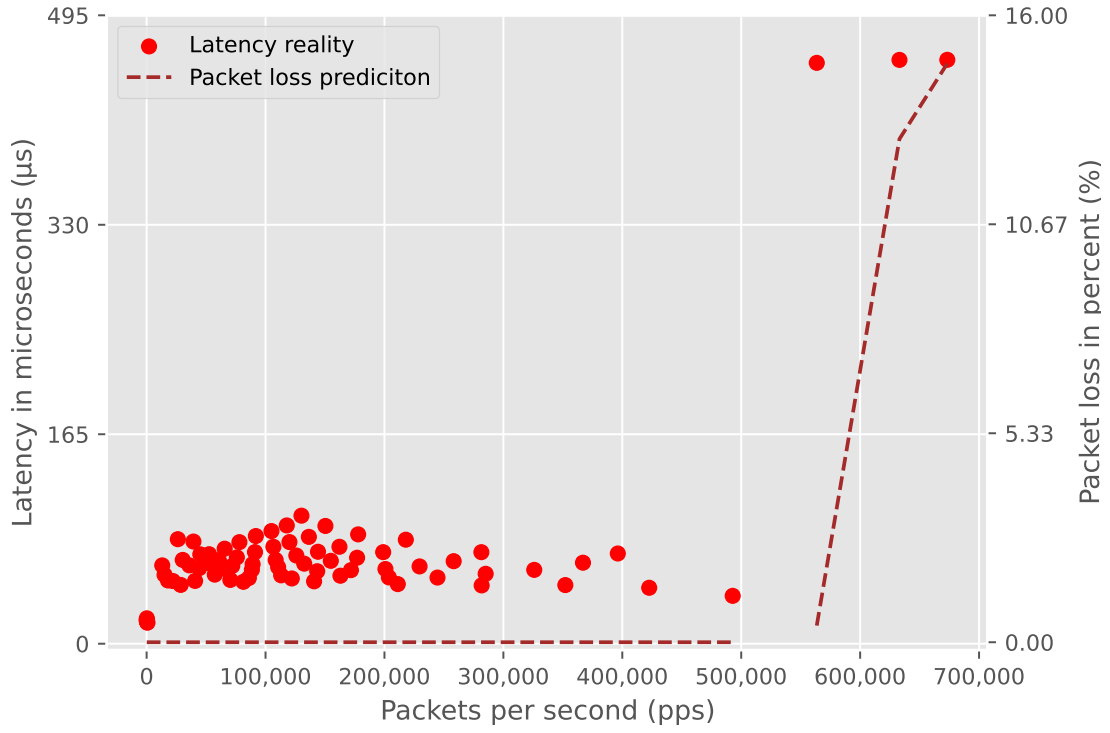


Figure 6.46: Packet loss prediction compared to the basic measurement on the DUT 1

6.5.7 Automation

To generate a suitable model for the DUT, several measurements are necessary. We test a large number of combinations to map the behavior of the device. We choose the measurement configurations in such a way that they represent the latency and packet loss behavior as well as possible. At the same time, the measurement does not require too much time. To avoid starting these measurements manually, we write multiple scripts in the programming language Python that perform the necessary

measurements. The scripts configure the DUTs and start the measurement. We define configuring the DUT as generating firewall rules on the device and setting the correct command line arguments for the packet generator. Depending on the DUT, the configuration is done via SSH or serial connection.

After all, measurements are done, we use the results to calculate important values such as standard deviation, mean value, maximum value, and minimum value. We store the resulting values in a summary csv file. The summary includes the number of firewall rules that we configure and the packet loss that we observe. We use this summary for our modeling application to find functions describing the behavior of the DUT.

6.5.8 Evaluation Scenario

Our goal is to create a model for predicting the latency behavior and packet loss of firewalls. To test our model for accuracy and robustness, we create an automatic test scenario. Our automatic test uses values that we did not generate in our profiling and use them as input for our model. Our model then generates predictions for the respective input. With the same settings, we measure the latency and packet loss at our DUT. The difference between the prediction of our model and the actual measured values determines the accuracy of our model. To keep the measurement error as small as possible, we repeat the latency and packet loss measurement on the DUT three times.

6.5.9 Precision

In the previous sections, we report on our measurements and their results. However, in any measurement, there can be inaccuracies or anomalies. In this section, we explicitly address the accuracy of our model and our latency measurements.

Since we only test a few firewall configurations in our profiling, we interpolate between the measurement results. By interpolating, we optimize the time required for our modeling. However, we increase the measurement error with this procedure. Every measurement contains a certain degree of measurement inaccuracy. To describe this in more detail, we show several measurements with the same configuration in Figure 6.47. Each triangle corresponds to an overload threshold that we measure. The illustration shows how the results differ, especially in the range of 0 - 100 firewall rules. With 0 firewall rules, one measurement point is at $\approx 633,000$ pps and two measurement points are at $\approx 580,000$ pps. With 1000 firewall rules, all three measurement thresholds are within $72,000$ pps and $72,080$ pps.

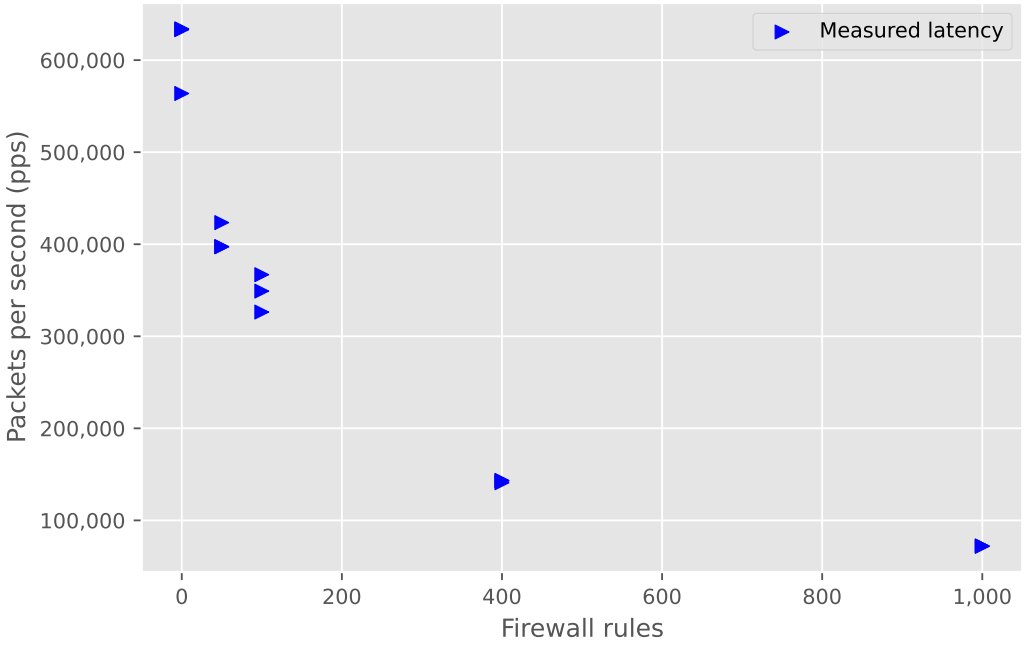


Figure 6.47: Multiple threshold measurement of the firewall configuration

We average the results from three measurements to reduce the effects of the measurement error in our measurements. We then use the mean values for our further calculation. This ensures that we reduce the measurement error and that further calculation does not make the error too large.

As already mentioned, we measure not only the latency but also the packet loss. Like the latency, we write the packet loss in a csv file. Since we do not assign an ID to each packet, we cannot determine exactly which packets get lost. However, we count the number of packets we generate and those we analyze. In Figure 5.2, we show the flow of those packets as orange arrows. The difference between them represents the packet loss. Inaccuracies can occur with this type of packet loss determination. The packets that we generate right before the measurement stops might not be analyzed by the generator because the measurement has already stopped. Another error occurs due to our warm-up phase. We do not measure the latency or packet loss during that time, but the generated packets still count to the overall generated and analyzed packet counter from MoonGen. Some packets that we generate just before the measurement starts but analyze during the measurement lead to a wrong number of generated and analyzed packets. We need additional information such as IDs in the packets to prevent these measurement errors.

7 Results and Discussion

In this chapter, we present and discuss our results. First, we show the results for the behavior prediction for different DUTs with our model. We classify our results and discuss how they correspond to our expectations. Our expectation is a model that we can apply to various software firewalls. We expect our model to predict the behavior of different industrial network traffic scenarios. Second, we describe the application areas of our model and how well it fits. Third, we show the limitations of our model. Afterward, we discuss the measurement limitations. Finally, we suggest improvements for iptables.

7.1 Prediction

We test our model to ensure that it is robust and applicable to different software firewalls. With the help of different network configurations, we can test our model. Therefore, we train our model with one set of network traffic scenarios.

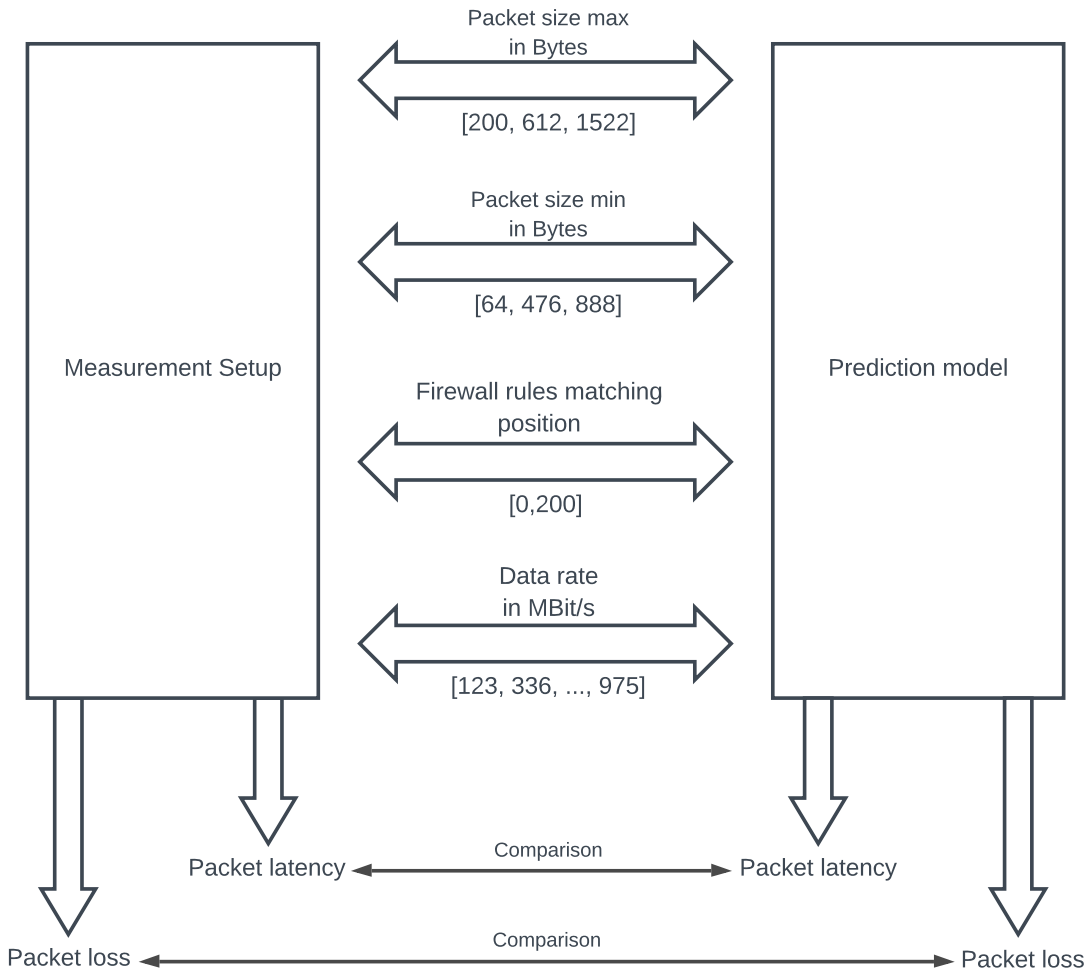


Figure 7.1: Verification setup

In the next step, we test our model with a different set of network traffic scenarios. We show the schematic structure for the verification measurements in Figure 7.1. In our verification measurement, we test configurations that our model does not already know through profiling. As described in Section 6.5.4, we specify three groups of traffic: shorter, medium, and larger frames. For this, we choose as minimum packet size 64 Bytes, 476 Bytes, and 888 Bytes. For the maximum limits, we choose 200 Bytes, 612 Bytes, and 1522 Bytes. We also define the data rate and the average firewall rule matching position. We vary the data rate between 123 MBit/s and 975 MBit/s. We measure the behavior at 0 firewall rules and the average firewall rule matching position at 200. Our verification only contains two rule configurations since we already test five firewall rule configurations in our profiling in Section 5.6. First, we test without firewall rules to ensure that our model reflects the base behavior. The base behavior is the behavior unaffected by firewall rule configurations or other options. We chose the 200 firewall rule configuration because it does not appear in our profiling. In addition, we assume, based on our profiling, that 200 firewall rules

already have an impact on the latency and packet loss behavior of firewalls. If there are significantly more than 200 firewall rules (e.g., more than 2000 rules), a DUT may not support this number.

In order to assess the verification predictions of our model, we also measure the network configurations we provide as input to our model on our DUT. By comparing the two results, we determine the accuracy of our model. To optimize our predictions, we apply our optimized latency prediction with additional network traffic knowledge for each DUT in an example. First, we show the results of our verification measurement on the EAGLE40 with iptables (DUT 1). Second, we show the behavior of the EAGLE30 (DUT 2) and how our model represents the behavior. Finally, we discuss our VPP firewall (DUT 3) model prediction. See Table 5.2, for a definition of the devices.

7.1.1 EAGLE40 Iptables

We demonstrate the entire implementation detailed in Chapter 6 using the DUT 1. In this section, we discuss the results of our verification measurement.

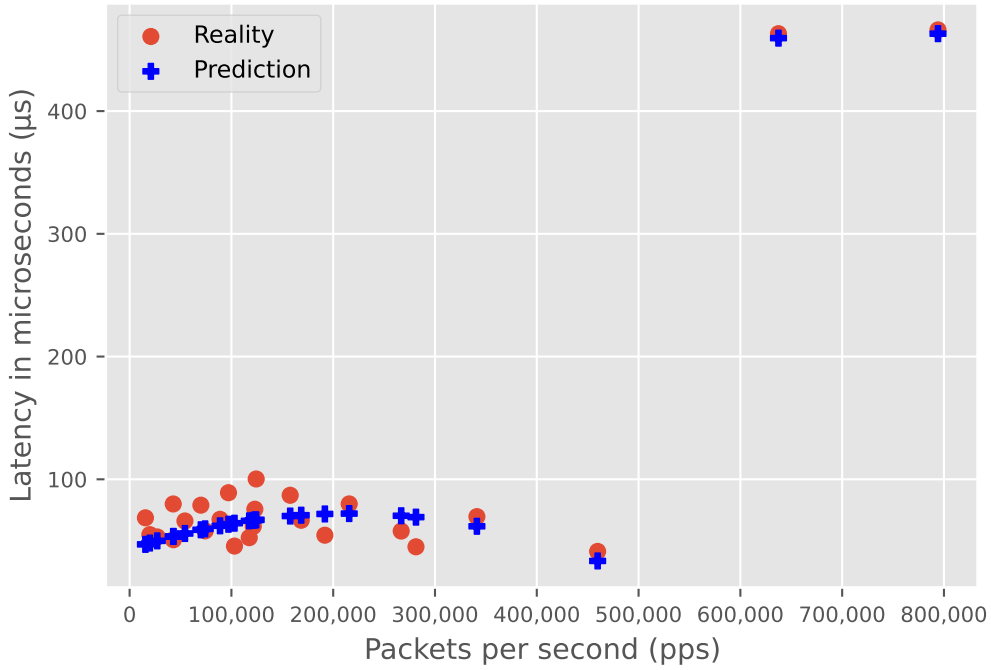
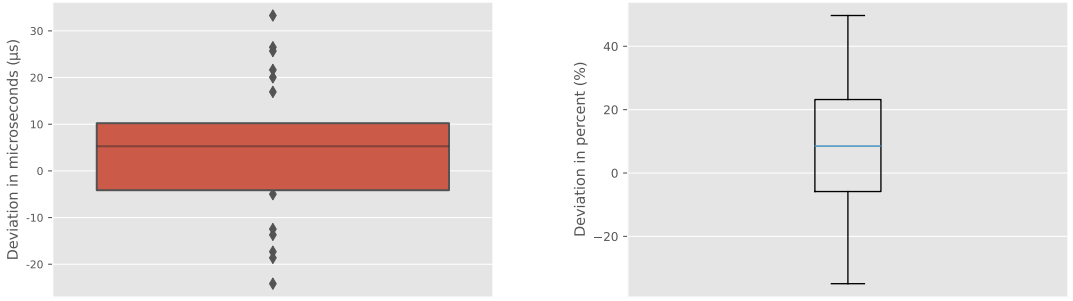


Figure 7.2: Comparison between DUT 1 verification measurement results and our model prediction functions based on our profiling; Latency in microseconds; 25 verification measurement results between 15,537.5 pps and 794,135.5 pps; The red points originate from our verification measurement without firewall rules

Figure 7.2 shows our comparison between the latency of our verification measurement and the latency prediction of our model. For this verification measurement, we

did not configure any firewall rules. The non-overload area is from 0 to ≈ 470,000 pps with latencies of less than 100 μs. The overload area starts from ≈ 630,000 pps with latencies of more than 450 μs. As we define in Section 6.1.1, the overload area is where packet loss occurs and latency increases.



(a) Deviation in microseconds

(b) Deviation in percent

Figure 7.3: Deviation between our packet latency prediction and the actual packet latency of the DUT 1 in microseconds and percent; We calculate the deviation from our prediction and the verification measurement without firewall rules

The comparison shows that our model prediction is close to the actual latency. For a more detailed comparison of the deviations, we show the deviations in microseconds in a letter-value plot in Figure 7.3(a) and the deviations in percent in Figure 7.3(b). The whiskers in Figure 7.3(b) span 1.5 times the IQR from the box. Our model has an average deviation on all predictions of 4.35 μs or 8.21 % from the latencies of the verification measurement. The confidence interval with a confidence level of 95 % is between -1.44 μs and 10.15 μs. Thus, the true mean value is in this range with a probability of 95 %. This means that our model underestimates the latency by 4.35 μs on average. Our model is, therefore, 8.21 % too optimistic in terms of latency prediction. The standard deviation for this prediction is 14.49 μs. In the worst case, our model is 33.28 μs below the real latency.

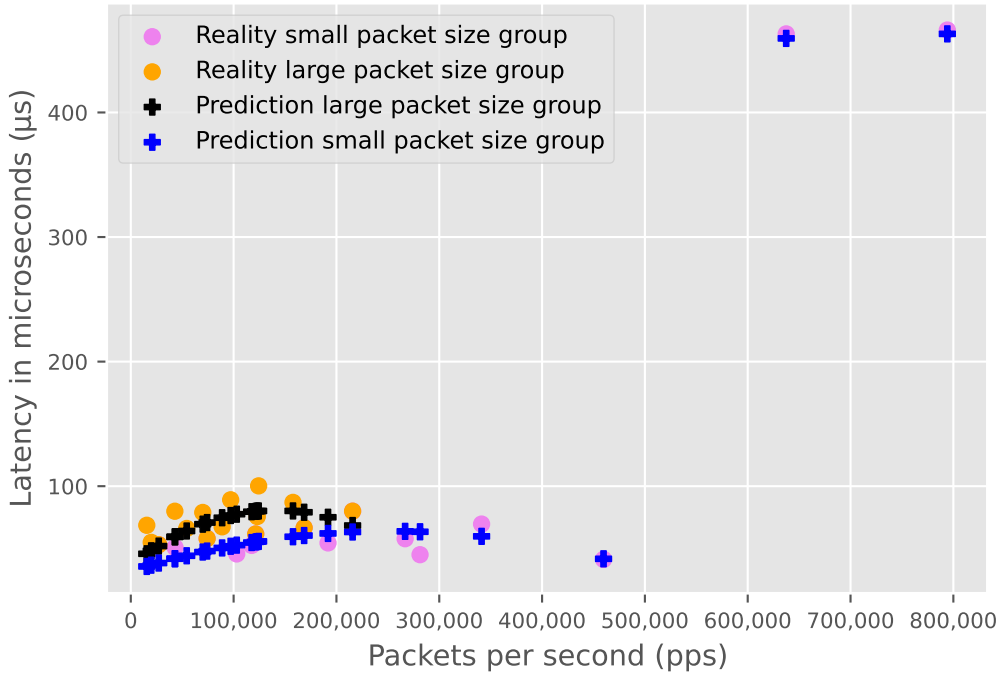
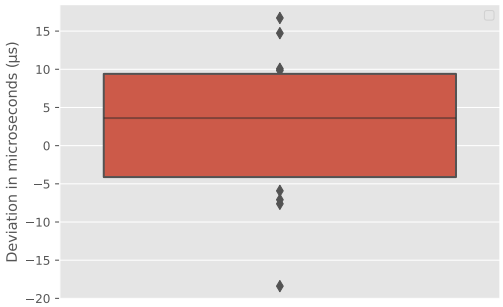
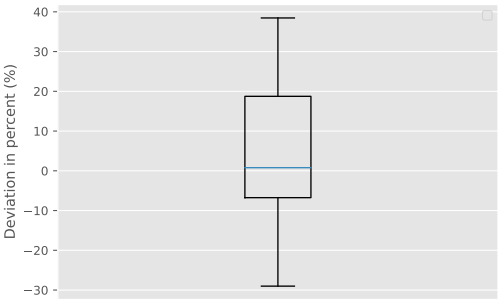


Figure 7.4: Comparison between DUT 1 verification measurement results and our optimized model prediction functions based on our profiling; Latency in microseconds; 25 verification measurement results between 15,537.5 pps and 794,135.5 pps; The orange and violet points originate from our verification measurement without firewall rules

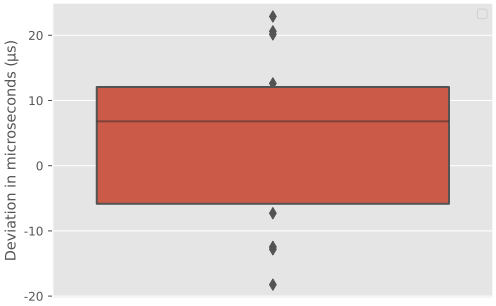
After we demonstrated our optimization in Section 6.5.4, we will present the results here. In Figure 7.4, we compare the results between our optimized latency prediction and the latencies we measure. We observe that we produce two different latency predictions. Once the prediction for large packet sizes and once the prediction for small packet sizes. The two predictions together now cover the latencies we measured better. We are modeling the optimization to predict the latencies more accurately and thus better quantify the real-time traffic capability.



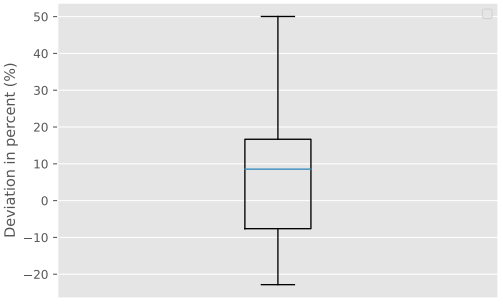
(a) Small packets deviation in microseconds



(b) Small packets deviation in percent



(c) Large packets deviation in microseconds



(d) Large packets deviation in percent

Figure 7.5: Deviation between our optimized packet latency prediction and the actual packet latency of the DUT 1 in microseconds and percent; We calculate the deviation from our prediction and the verification measurement without firewall rules; 25 verification measurement results between $15,537.5$ pps and $794,135.5$ pps; Top left and right represent the deviation for small packet sizes; Bottom left and right represent the deviation for large packet sizes

We show the deviations between our optimized prediction and the measured latencies in Figure 7.5. The Figure 7.5(a) shows the deviation for small packet sizes in microseconds. The Figure 7.5(b) shows the deviation for small packet sizes in percent. The average deviation for small packet sizes is 2.53 μs respectively 5.18 % after our optimization. Our standard deviation, in this case, is 9.19 μs. The maximum deviation for small packets is 18.39 μs. In Figure 7.5(a) and Figure 7.5(b), we display the deviations for the large packet sizes. On average, the deviation for large packet sizes is 3.91 μs respectively 7.52 % after our optimization. The standard deviation, in this case, is 7.52 μs. The maximum deviation for small packets is 22.88 μs. We thus show that our optimization has lower latency deviations than our non-optimized prediction. With the optimization, we can more accurately quantify the real-time capability of the DUT. The latencies of the DUT 1 in the non-overload area are sufficient for cyclic traffic.

In another verification measurement, we show how our model responds to firewall

rule configurations. As we previously discussed, we configure the matching firewall rule at position 200. We aim to show the adaption of our model to a new network traffic scenario.

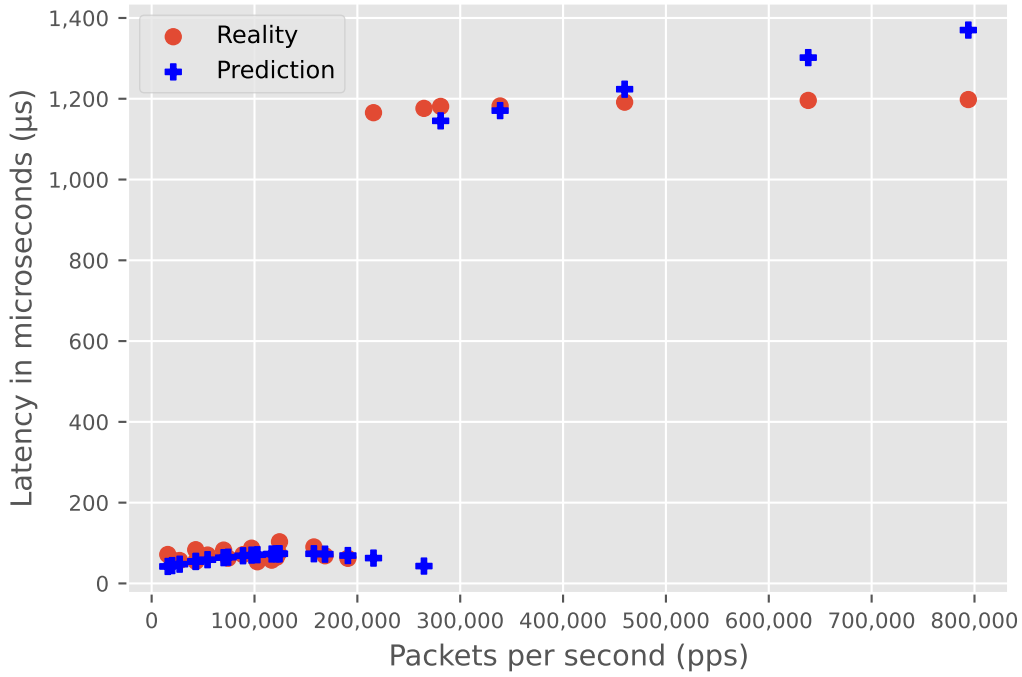
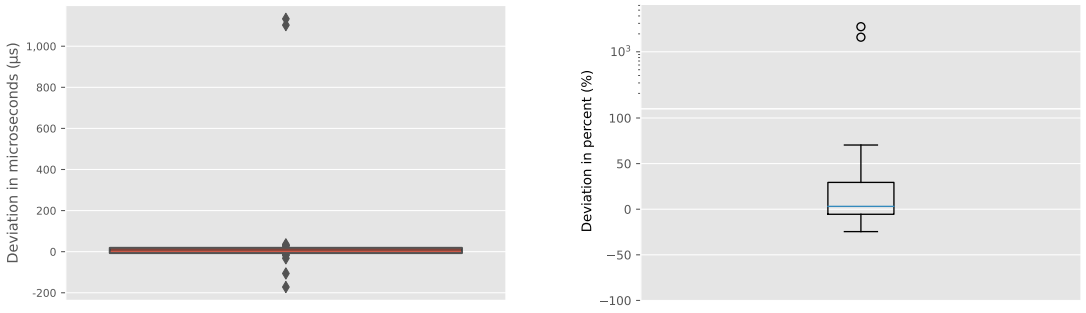


Figure 7.6: Comparison between measurement results and our model prediction; DUT 1 with 200 firewall rules configured; Latency in microseconds; 25 verification measurement results between 15,683.5 pps and 794,006 pps; The red points originate from our verification measurement with 200 firewall rules

In Figure 7.6, we compare the results between predicted latency and the latency we measure. At 215,694 pps and 264,756.5 pps we notice that the latency prediction and the measured latency differ significantly. At 221,756.5 pps our model predicts 62.73 μs, but we measure 1,165.50 μs. In addition, our model predicts 42.96 μs at 264,756.5 pps, but we measure 1,176.18 μs. These deviations in the overload area are not relevant for the quality of our model. Our model is intended for the prediction of devices in an industrial network. In these networks, a device must never operate in the overload area, otherwise packet loss and extreme latencies will occur. Therefore, a misinterpretation of the overload threshold is not relevant for the quality of our model.

We observe that our model predicts the overload area about 50,000 pps too late. The non-accurate prediction of the threshold causes the two huge deviations between latency prediction and measurement. Our model predicts the threshold for DUT 1 with 200 firewall rules at $\approx 260,000$ pps. In our measurement, we observe that the threshold for DUT 1 with 200 firewall rules already starts at $\approx 210,000$ pps. The

shifted threshold is due to our threshold prediction function. As previously shown, in Figure 6.14, our function predicts the threshold at $\approx 260,000$ pps. The deviation occurs because the curve fit function tries to minimize the distance to all threshold data points we measure. This results in the deviation of our threshold prediction function from the values we measure.

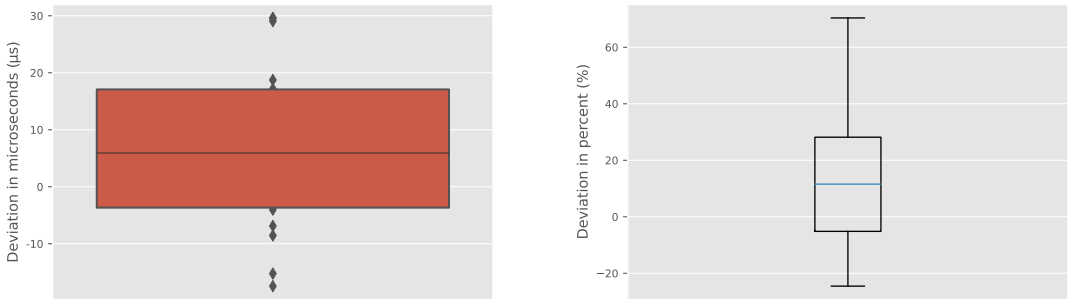


(a) Deviation in microseconds

(b) Deviation in percent

Figure 7.7: Deviation between our packet latency prediction with a firewall matching position at 200 and the actual packet latency of the DUT 1 in microseconds and percent; We calculate the deviation from our prediction and the verification measurement with 200 firewall rules

The outliers between our prediction and our measured latency are also visible in Figure 7.7. We show in Figure 7.7(a) and Figure 7.7(b) how we observe the deviations in microseconds and percent. Due to the wrong threshold, the mean deviation of our prediction is 83.95 μs, which corresponds to 184.73 %. The true mean value is between -39.27 μs and 207.19 μs with a five percent error probability. Our model tends to predict the latencies too low. In this case, our model can predict the latencies with an average inaccuracy of 184.73 %. The standard deviation for our latency prediction with 200 firewall rules is 308.02 μs. In the worst case, our model is $1,133.21$ μs below the real latency. The high standard deviation and the high mean value result from the incorrect predictions in the transition range between non-overload and overload area. In Figure 7.7(b), we show that the two outliers deviate significantly from our other predictions. The reason for the high deviation is that our model does not determine the threshold correctly. The threshold determination function in Equation (6.2) can lead to deviations from the real thresholds.

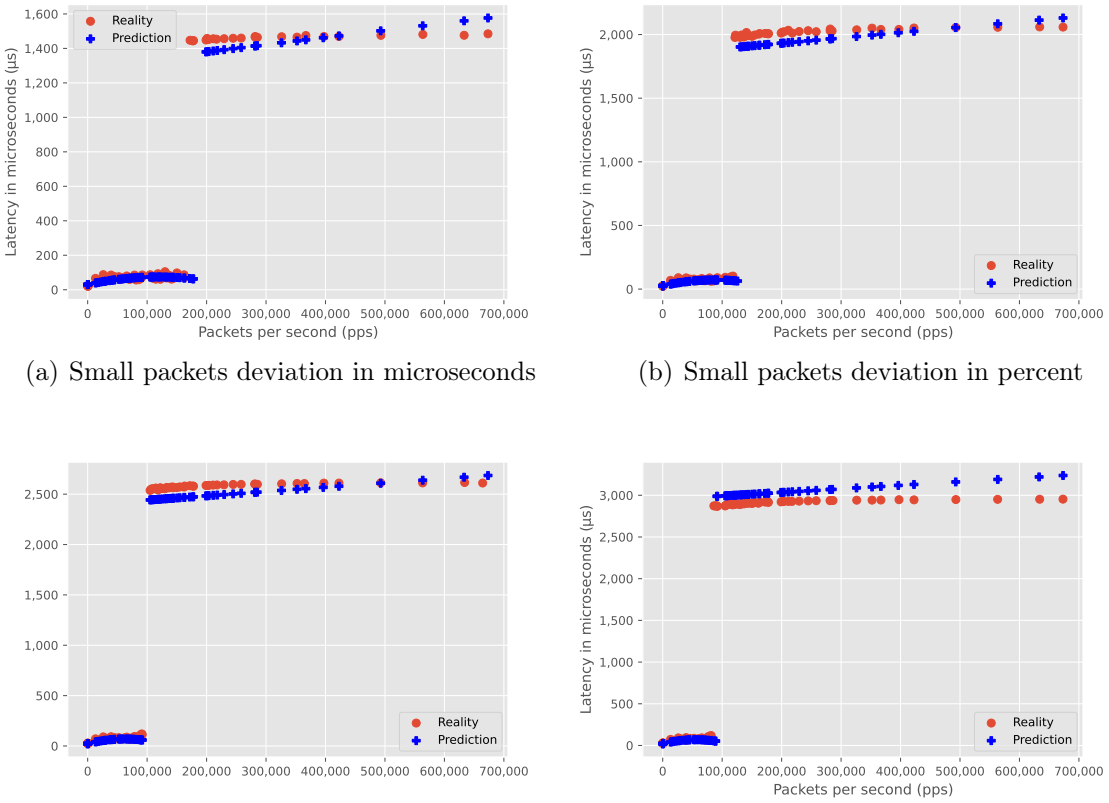


(a) Deviation in microseconds

(b) Deviation in percent

Figure 7.8: Deviation between our packet latency prediction with a firewall matching position at 200 and the actual packet latency of the DUT 1 in microseconds and percent; Only for the non-overload area; We calculate the deviation from our prediction and the verification measurement with 200 firewall rules

Since we have designed our model for firewalls in industrial networks, we now consider only the non-overload area since this is the only one that occurs in industrial networks. In Figure 7.8, we show the deviation only for the non-overload area between our prediction and the measured latency at 200 firewall rules. We observe that the latency deviations in Figure 7.8(a) are smaller than in Figure 7.7(a). The same applies to the percentage deviation in Figure 7.8(b). Looking only at the prediction results for the non-overload area, our model has an average deviation of 6.99 μs, which corresponds to 12.45 %. The true mean value is between 0.13 μs and 13.85 μs with a five percent error probability. Our standard deviation, in this case, is 14.43 μs. The maximum deviation is then only 29.61 μs. We show that our model also accurately predicts the configuration of firewall rules for the non-overload area, which is essential in our use case.



(a) Small packets deviation in microseconds

(b) Small packets deviation in percent

(c) Large packets deviation in microseconds

(d) Large packets deviation in percent

Figure 7.9: Comparison between verification measurement results and our model prediction for DUT 1; Latency in microseconds; Top left 300 firewall rules; Top right 500 firewall rules; Bottom left 700 firewall rules; Bottom right 900 firewall rules

In Figure 7.9, we compare four more predictions with measurements for the DUT 1, each with different network settings. Here, we demonstrate that our model can predict multiple firewall rule configurations. The Figure 7.9(a), shows the comparison between our latency prediction and the latency we measure with 300 firewall rules. We notice that here, just like in Figure 7.6, our model predicts the overload range too late at 200,000 pps. Therefore, our model predicts a few points between 180,000 pps and 200,000 pps with too low latency. The reason for this is, as in Figure 7.6, the deviation of our threshold function from reality.

The Figure 7.9(b) shows our latency prediction in comparison with the latency we measure for 500 firewall rules. With 500 firewall rules, there is still a slight deviation between the predicted threshold and the one we observe in our measurement. But we notice that the deviation our threshold prediction generates is getting smaller.

The Figure 7.9(c), shows the comparison between our latency prediction and the latency we measure at 700 firewall rules. In this network scenario, our threshold prediction is correct. As a result, our model does not produce extremely divergent latency predictions.

The Figure 7.9(d), compares our latency prediction and the latency we measure

at 900 firewall rules. In this diagram, we notice that above 300,000 pps the latency of our prediction is higher than the latency we measure. This is due to the linear increase of the prediction function in our model, but the measured latency does not have the same slope.

7.1.2 EAGLE30 Iptables

In this section, we show our results for the EAGLE30 (DUT 2) and discuss them. Our goal is to show that our model can be applied to other firewalls as well. First, we profile the DUT 2 with our profiling steps that we describe in Section 5.6. We use the results of our profiling to create a model for the DUT 2. In Section 6.5 we describe the general construction of our model.

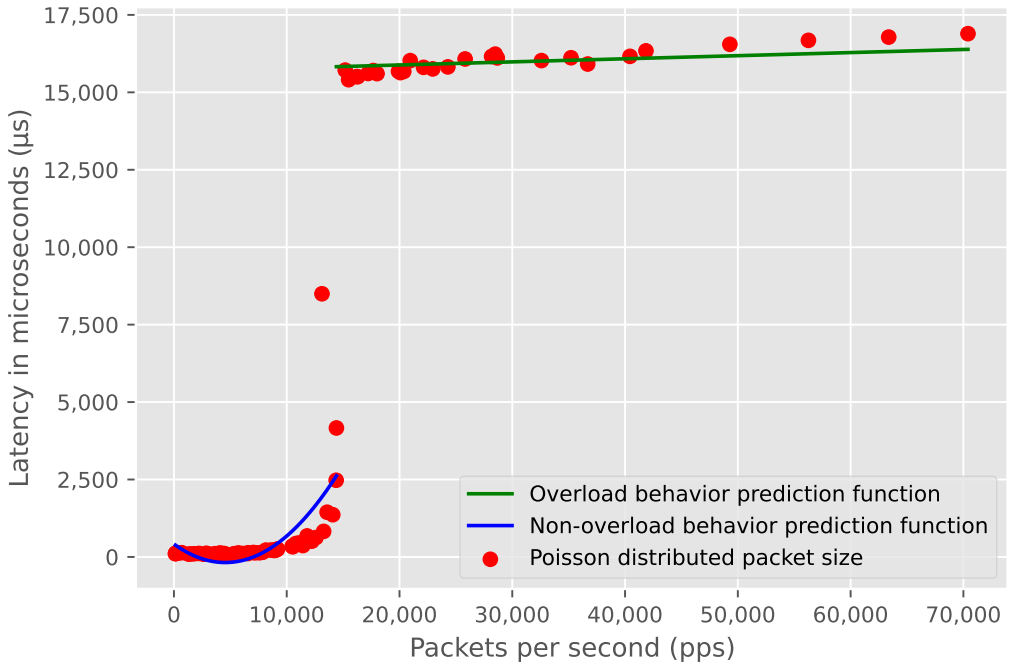


Figure 7.10: Comparison between DUT 2 profiling measurement results and our model prediction functions based on our profiling; The red points originate from our profiling measurement without firewall rules

Figure 7.10 shows the resulting model for the DUT 2. The model, in Figure 7.10, represents the behavior of the DUT 2 without firewall rules. We notice that our profiling measures a maximum of 70,000 pps. This is because the DUT 2 has a maximum data rate of 100 MBit/s on the RJ45 interfaces. We observe that the DUT 2 behaves similarly to the DUT 1 in Figure 6.37. Just like the DUT 1, the DUT 2 also has a non-overload area and an overload area. The overload area of the DUT 2 starts earlier and has a higher latency than the overload area of the DUT 1. These findings show that iptables firewalls behave similarly. The hardware

influences the latency of the different iptables firewalls. The better the hardware performance, the lower the packet latency.

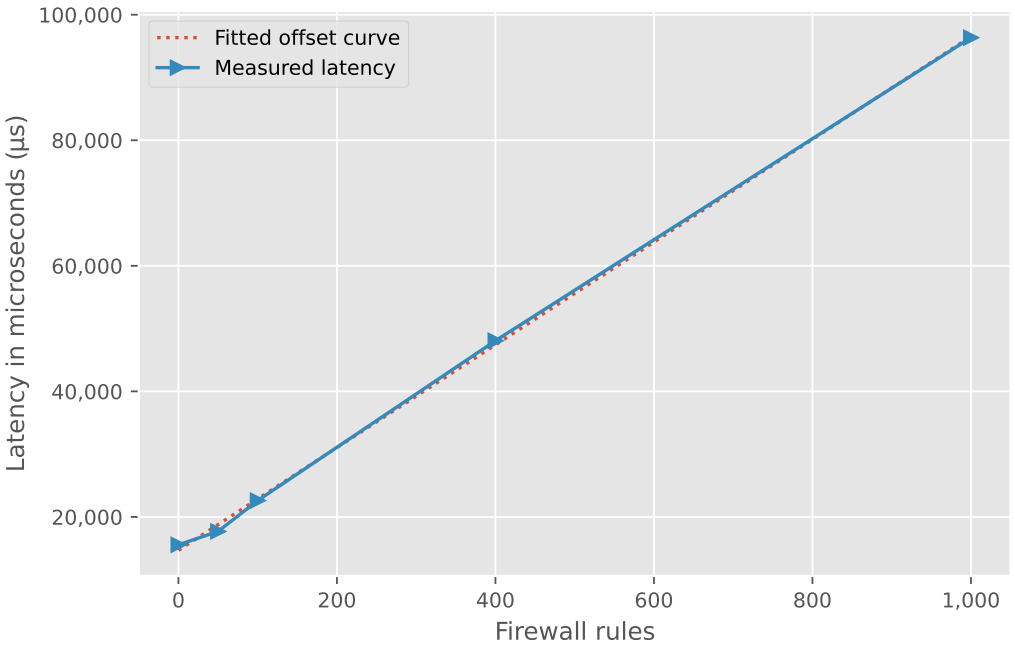


Figure 7.11: Overload are maximum latency prediction curve and latency we observe through our profiling for the DUT 2

In terms of firewall dependency, we expect a linear increase in latency with an increasing number of firewall rules. In Figure 7.11, we show the maximum packet latencies that we measure with our profiling for the different firewall rule configurations. We show that the latency behaves as we expect it. The more firewall rules the DUT 2 has to consider, the higher the packet latency. The latency increases linearly with the number of firewall rules that we configure, from $15,559.08\ \mu s$ at 0 firewall rules to $96,343.11\ \mu s$ at 1000 firewall rules. In comparison to the maximum latency prediction function in Figure 6.16, we observe that the latency of the DUT 2 is consistently higher than the DUT 1 latency.

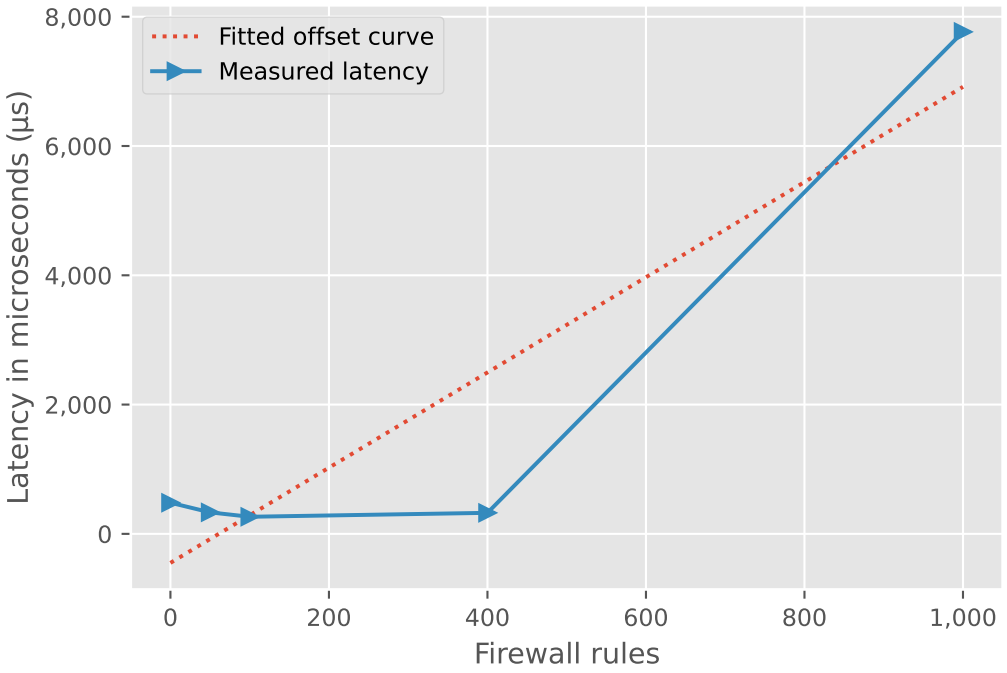


Figure 7.12: Non-overload area latency prediction curve and latency we observe through our profiling for the DUT 2

For the non-overload area of the DUT 2, we observe similar behavior in comparison to the DUT 1. In Figure 7.12, we show the mean latency of the non-overload area for different firewall configurations. Again, the basic latency increases with the number of firewall rules. We measure 481.72 μs with 0 firewall rules and 7765.64 μs with 1000 firewall rules.

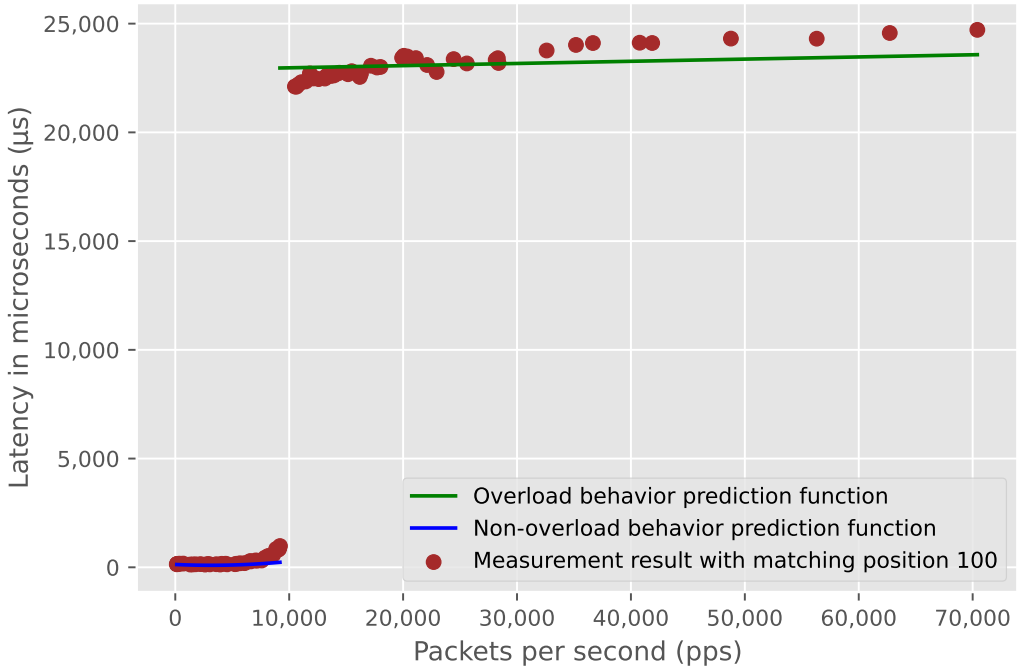


Figure 7.13: Comparison between DUT 2 latency behavior with 100 firewall rules and our DUT 2 model latency behavior prediction functions for 100 firewall rules; The brown points originate from our profiling measurement with 100 firewall rules

A feature of our model is the prediction of the latency with different firewall configurations. To verify this, we use our model to generate a latency prediction for 100 firewall rules on the DUT 2. We show the latency comparison between our latency prediction and our latency measurement in Figure 7.13. We detect the non-overload area from 0 pps to 10,000 pps and the overload area above 10,000 pps. Using our maximum latency prediction from Figure 7.11, we can determine the latency offset for the overload area. Our model adjusts the prediction functions accordingly. With this method, our model can predict the behavior of different firewall rule configurations. With a configuration of 100 firewall rules, our model has an average deviation of 76.17 μs. In this case, our model tends to under-predict the latency. The standard deviation is 442.84 μs. The high deviation comes from the fact that the latencies of the DUT 2 are very high in absolute terms. Small percentage deviations already lead to high latency deviations. If we keep this in mind, our model represents the behavior of the DUT 2 well. The DUT 2 with 100 firewall rules is only suitable for industrial networks up to $\approx 10,000$ pps. Above $\approx 10,000$ pps, packet loss occurs and the latencies are too high for cyclic traffic.

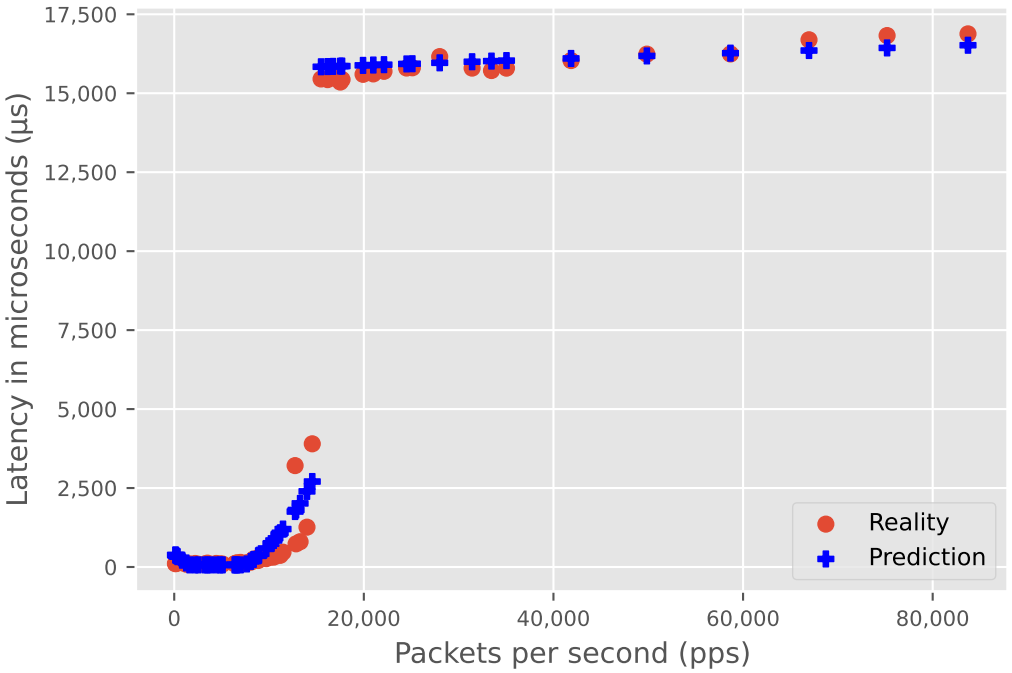
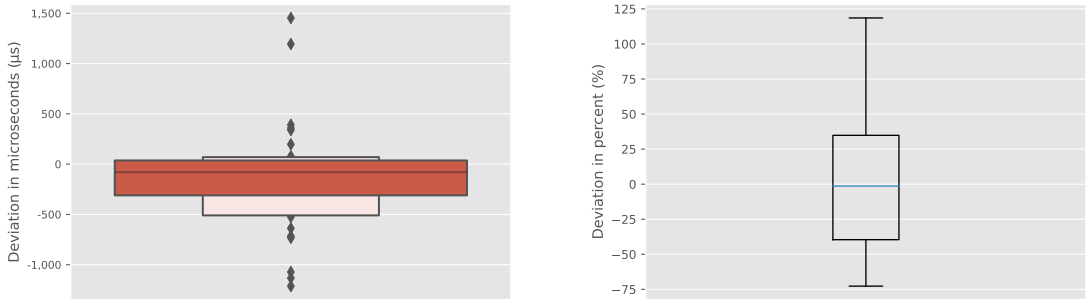


Figure 7.14: Comparison between DUT 2 verification measurement results without firewall rules and our model prediction functions; Latency in microseconds; The red points originate from our verification measurement without firewall rules

In Section 6.5.8, we discuss how to verify our model to determine its accuracy and robustness. Accordingly, we generate a verification measurement to test our DUTs. Here, we discuss the results from the verification of our model for the DUT 2. In Figure 7.14, we compare the latency prediction of our model and the latencies we measure with 0 firewall rules in our verification measurement. In comparison, Figure 7.10 shows the measurement results from our profiling compared with the model prediction. To verify our DUT 2 model, we measures 55 pps values between 127.5 pps and 83, 724 pps. We observe that our prediction works very well, especially in the overload area. Due to the curve fitting of our model and the latency behavior of the DUT 2, there are some pps values where our latency prediction deviates more significantly from reality.



(a) Deviation in microseconds

(b) Deviation in percent

Figure 7.15: Deviation between our packet latency prediction and the actual packet latency of the DUT 2 in microseconds and percent; We calculate the deviation from our prediction and the verification measurement without firewall rules

To better illustrate the deviation between the latency prediction of our model and the actual latency, we draw Figure 7.15(a). Our mean deviation, in this case, is $-136.10\ \mu s$. With an error probability of five percent, the true mean value lies between $-252.62\ \mu s$ and $-19.59\ \mu s$. This means that, on average, we predict the latency to be $136.10\ \mu s$ too large. We show the percentage deviations in Figure 7.15(b). If we look at this deviation in percentage terms, our prediction is, on average, 0.38 % higher than the real latency. The standard deviation is $436.85\ \mu s$. The high standard deviation is due to the fact that the latencies of the DUT 2 are generally very high and small deviations in percentage terms result in a large latency in absolute terms. We demonstrate that our model can predict the latencies well without much effort. Our model shows that the DUT 2 could only handle a small number of pps in an industrial network since packet loss and high latency already occur at $\approx 17,000$ pps.

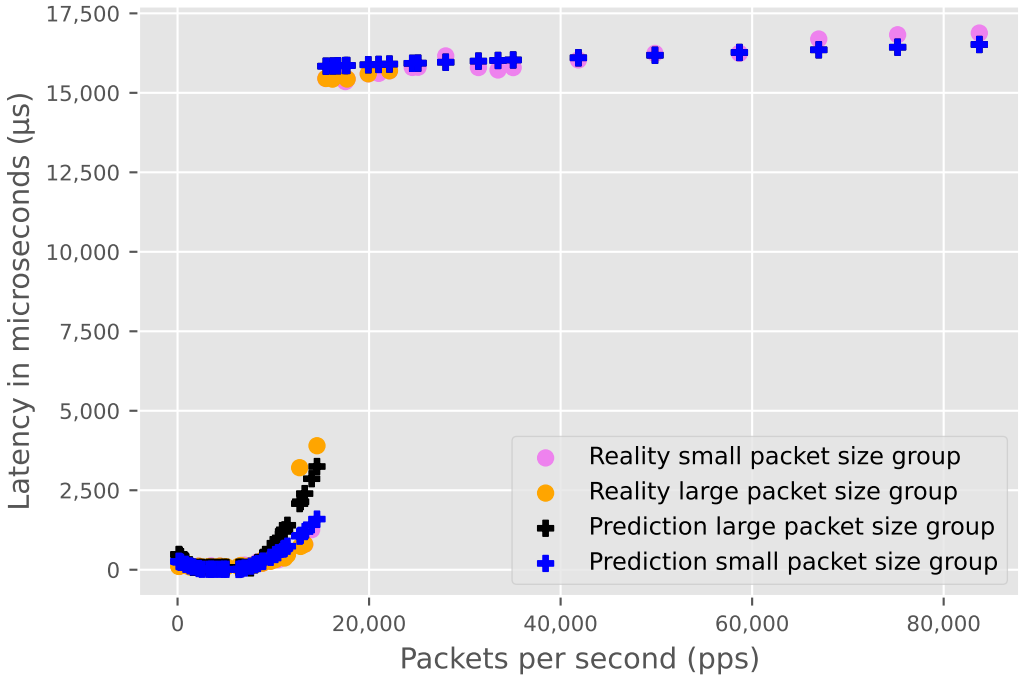
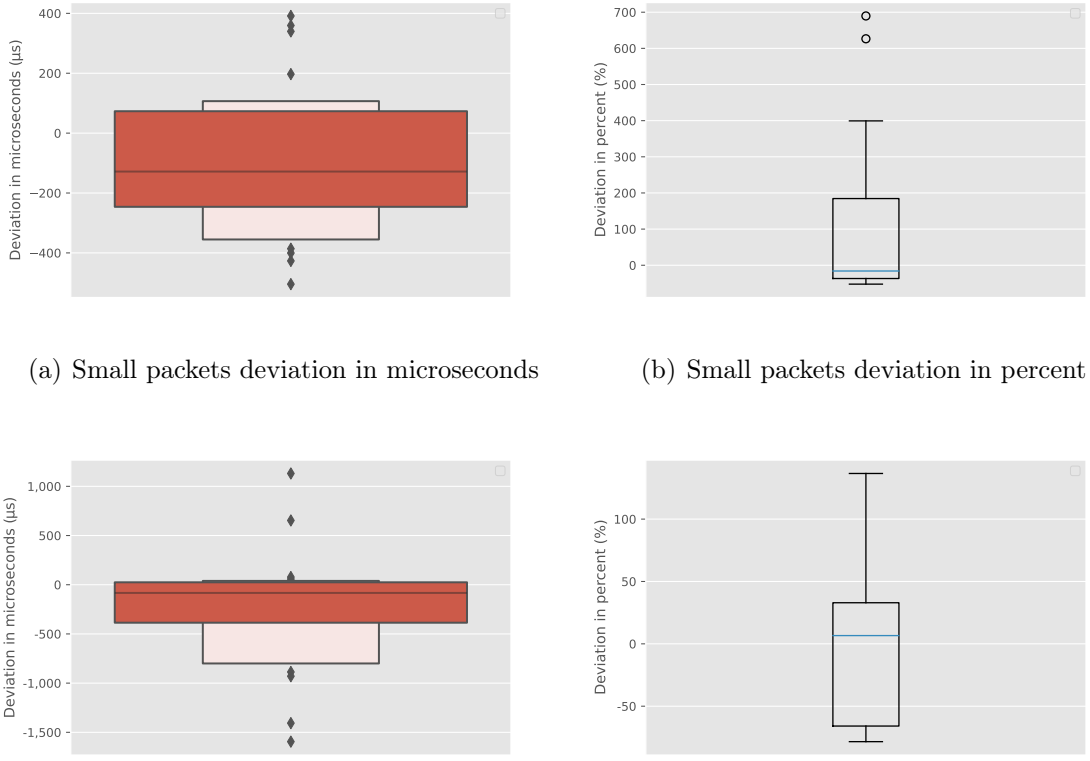


Figure 7.16: Comparison between DUT 2 optimized profiling measurement results and our model prediction in microseconds; Latency in microseconds; The orange and violet points originate from our verification measurement without firewall rules

In Figure 7.16, we consider the result of our optimized prediction for large and small packet sizes. We see that the two functions predict the appropriate latencies for the respective packet sizes. This improves the deviation of our latency prediction. The average deviation for small packets is now $-91.36\ \mu s$ and for large packets $-226.03\ \mu s$. The high deviation for large packets is due to the fact that the curve fit function also tries to find the smallest distance to the two points at $12,500$ pps and $14,400$ pps.



(a) Small packets deviation in microseconds

(b) Small packets deviation in percent

(c) Large packets deviation in microseconds

(d) Large packets deviation in percent

Figure 7.17: Deviation between our optimized packet latency prediction and the actual packet latency of the DUT 2 in microseconds and percent; We calculate the deviation from our prediction and the verification measurement without firewall rules; Top left and right represent the deviation for small packet sizes; Bottom left and right represent the deviation for large packet sizes

In Figure 7.17, we show the latency deviations from the optimized prediction. Through optimization, we achieve an average deviation of $-91.36\ \mu s$ or 55.31 % for small packets and $-226.03\ \mu s$ or -5.76 % for large packets. Due to the high latencies compared to the DUT 1, the optimization produces a too low latency prediction for small packets in the range between $2,000$ pps and $8,000$ pps. The low latency prediction results in a high percentage deviation. The curve fitting calculates the smallest deviation between all profiling data points and the fitted function. Since the latencies in the non-overload range of DUT 2 already vary between 100 μs and $4,000\ \mu s$, the curve fitting has to compromise. With the DUT 2, our optimization does not offer any improvement. Even large packets lead to overload and therefore our curve fitting has to make a trade-off. These extremely high latencies at already low pps are not applicable in a performance-optimized industrial network. Therefore, our optimization for the DUT 2 is good because it shows that even large packets lead to overload early on.

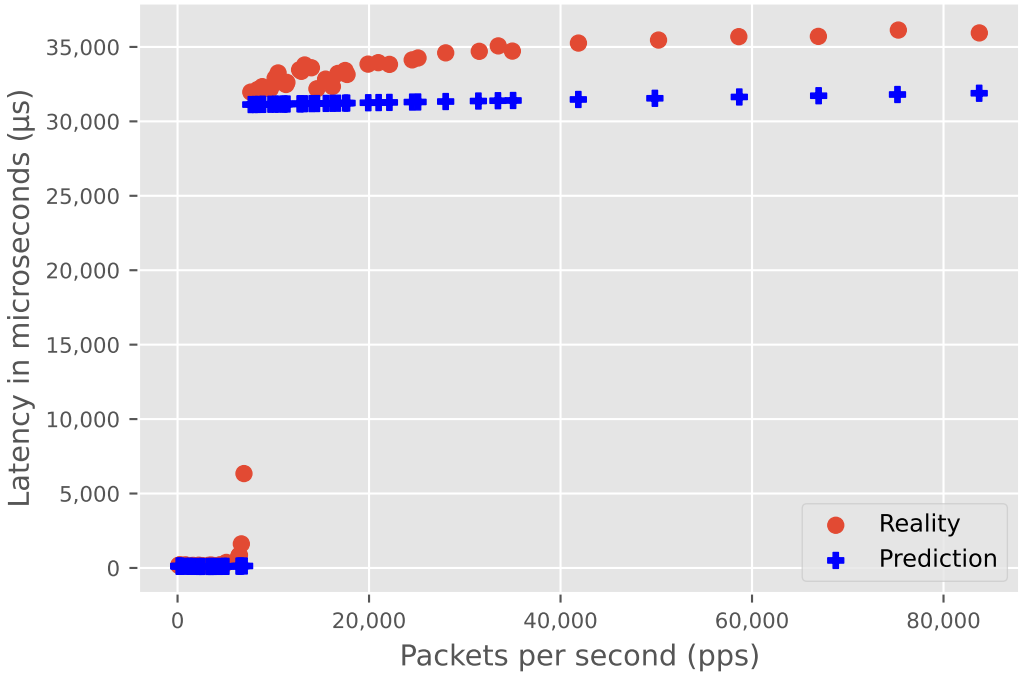
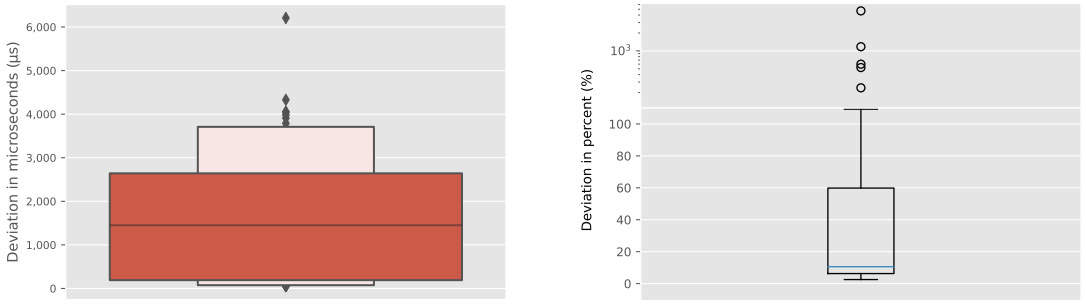


Figure 7.18: Comparison between DUT 2 (200 firewall rules) profiling measurement results and our model prediction in microseconds; Latency in microseconds; The red points originate from our verification measurement with 200 firewall rules

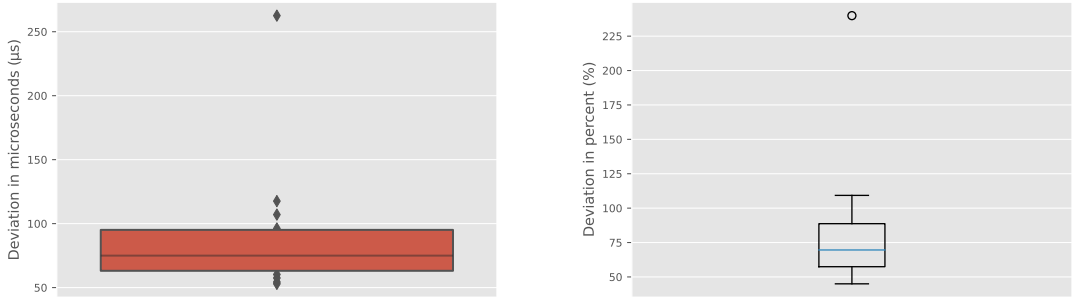
Figure 7.18 shows the latency we measure with our verification measurement on the DUT 2 and the latencies our model predicts for this scenario. In comparison to Figure 7.6, we observe that our model predicts the threshold more accurately for the DUT 2. However, the latency prediction differs especially in the overload area from the real latency. In industrial networks, firewalls are not operated in the overload area, as this would negatively affect the network performance. Therefore, the deviations in the overload area are not relevant for the quality of our model. Our deviation comes from the lower slope of the overload prediction function. The real latencies increase even more after the threshold. The function that our profiling generates has a lower slope, because the latencies we measured in the profiling do not show such a sharp increase in latency in the overload range. If we would generate an additional higher latency offset in our model, our prediction would be more conservative and in this case also closer to the measurement results. However, the additional latency offset would cause us to predict too high latencies in other network scenarios.



(a) Deviation in microseconds (b) Deviation in percent

Figure 7.19: Deviation between our packet latency prediction with a firewall matching position at 200 and the actual packet latency of an DUT 2 in microseconds and percent; We calculate the deviation from our prediction and the verification measurement with 200 firewall rules

Figure 7.19 presents the deviations in microseconds (Figure 7.19(a)) and percent (Figure 7.19(b)). The average deviation is $1,728.1\ \mu s$ or 154.56 %. With an error probability of five percent, the actual mean value is between $1,338.33\ \mu s$ and $2,118.02\ \mu s$. In Figure 7.19(a), we observe that most deviations are below $2,000\ \mu s$. In this network scenario, the latency deviations below 10,000 pps are mainly responsible for the high average deviation of 154.56 %. The largest deviation that our model shows occurs in the threshold range and amounts to $6,207.89\ \mu s$. In the overload area, we predict latencies too low, although they are significantly higher in our measurement. This is because we calculate through our profiling a lower slope in the overload area than we measure for 200 firewall rules.



(a) Deviation in microseconds (b) Deviation in percent

Figure 7.20: Deviation between our packet latency prediction with a firewall matching position at 200 and the actual packet latency of an DUT 2 in microseconds and percent; Only non-overload area; We calculate the deviation from our prediction and the verification measurement with 200 firewall rules

If we now consider only the non-overload range, we observe the deviations in Figure 7.20. The average deviation is now 89.77 μs or 82.45 %. With an error probability of five percent, the actual mean value is between 63.68 μs and 115.87 μs. The largest deviation, in this case, is 262.56 μs. We reduce the standard deviation to 49.81 μs. We thus show that we make better predictions in the non-overload range than over the entire pps range. It is precisely the non-overload area where our prediction should be more accurate, as this area is essential for industrial network suitability.

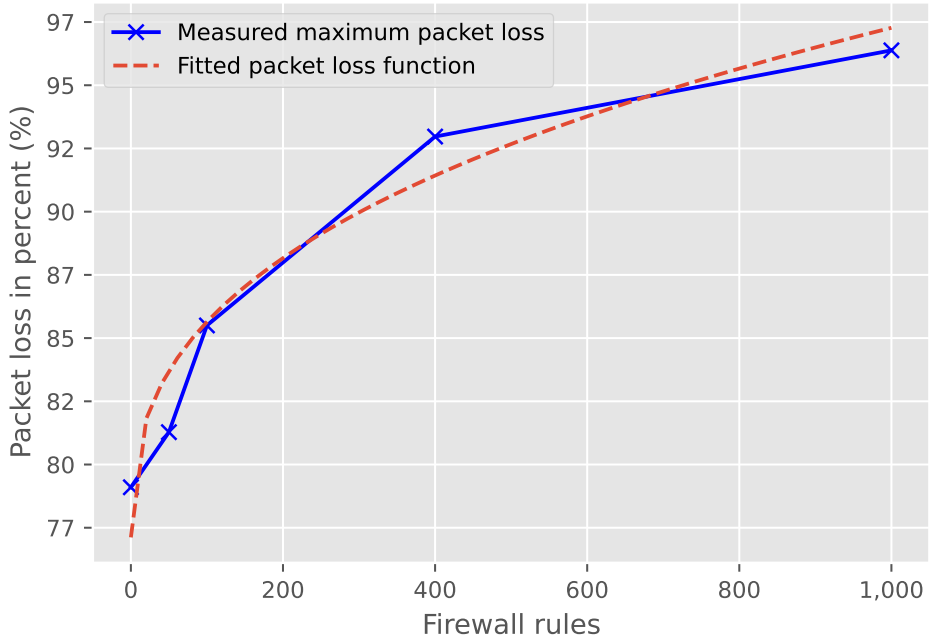


Figure 7.21: Packet loss determination for the DUT 2

So far, we determine the behavior of the latency with our model. In Section 6.5.6 we describe that our model can represent the packet loss as well. In Figure 7.21 we show that the maximum packet loss behaves as we describe in Section 6.5.6. From the root function shown in Figure 7.21, we calculate the vertex of our packet loss function for the DUT 2. With the knowledge of the threshold and the vertex, we calculate our packet loss curve in Figure 7.22.

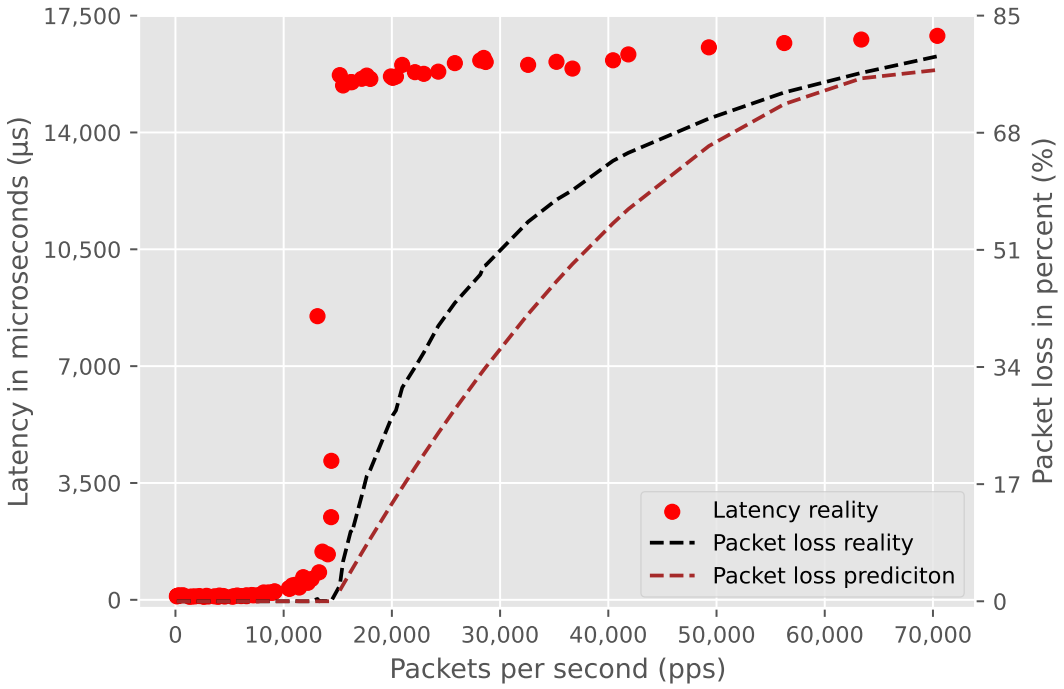


Figure 7.22: Comparison of our packet loss prediction and real packet loss for the DUT 2; The red points originate from our profiling measurement without firewall rules

With the DUT 2, we check how far the packet loss influences the overload area. We compare the real packet loss and our packet loss prediction in Figure 7.22. In comparison to the packet loss, we show the latencies we observe. Figure 7.22 shows that the overload area of the DUT 2 relates to the packet loss. The packet loss behavior of the DUT 2 is similar to the DUT 1 packet loss behavior. Our model accurately predicts the start of packet loss, which is essential for quantifying industrial network capability.

7.1.3 Vector Packet Processing (VPP)

In the previous sections, we applied our model to iptables software firewalls. In this section, we apply our model to the VPP software firewall. VPP „[...] is a modularized and extensible software framework for building bespoke network data plane applications.“ [49] Accordingly, there are different implementations that we test. First, we test how our model maps the behavior of VPP with single-core settings. Then, we test our model on VPP with multi-core settings.

7.1.3.1 VPP Single-Core

VPP with single-core settings enables the packet processing on a single CPU core. We dedicate this section to profile the behavior of the VPP firewall with the single-core setting.

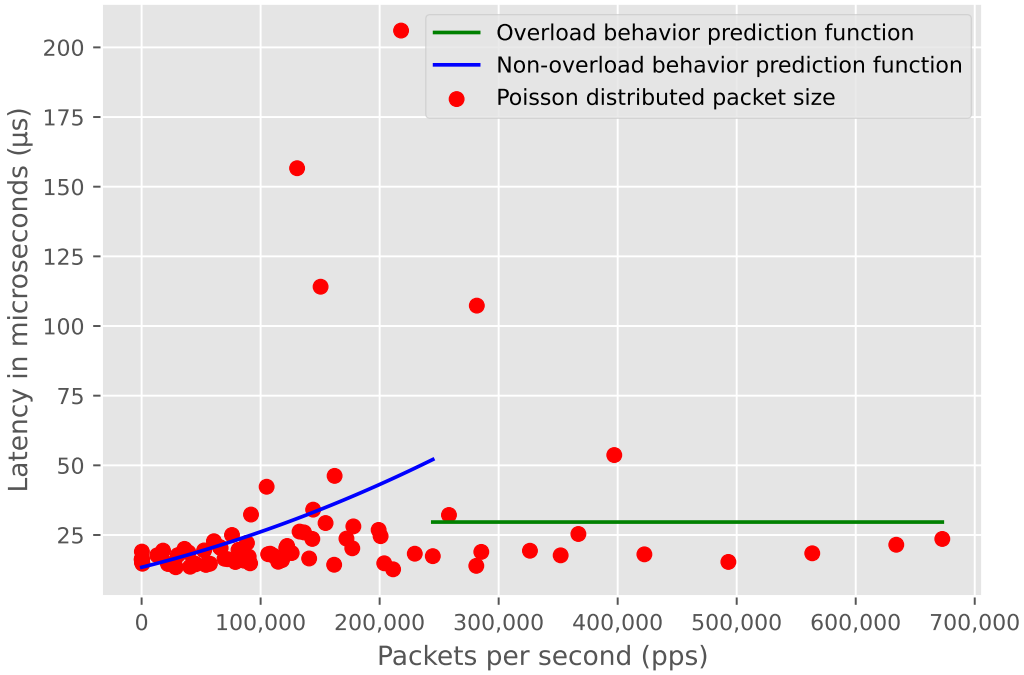


Figure 7.23: Comparison between VPP single-core verification measurement results and our model prediction functions based on our profiling; The red points originate from our profiling measurement without firewall rules

In Figure 7.23, we show the profiling result of how our model predicts the behavior of VPP without ACL rules. Although VPP does not behave in a typical overload manner like iptables firewalls, our model uses both the overload and non-overload prediction functions. The reason for this are the large latency deviations between $100,000$ pps and $300,000$ pps. Our model interprets these deviations as an overload threshold. In the single core setting without ACL rules, most of the latencies we measured are in the range of less than 50 μs. Compared to the profiling of DUT 1 in Section 7.1.1 and DUT 2 in Section 7.1.2, the VPP firewall does not exhibit overload behavior without ACL rules. VPP can process up to 256 packets simultaneously using vectorized packet processing. The vectorized packet processing is why the VPP firewall does not get directly into the overload area.

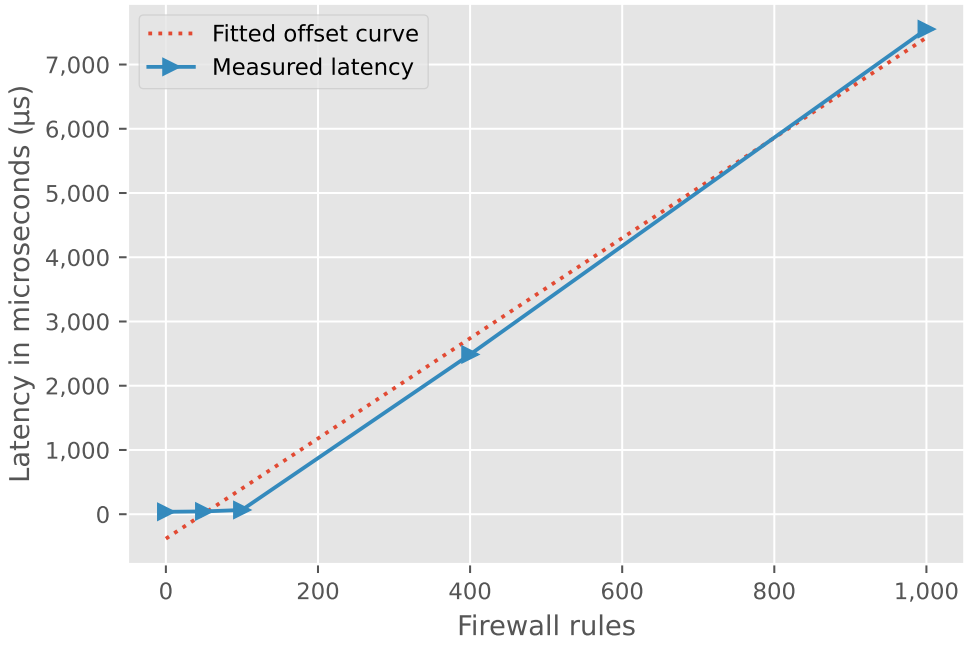


Figure 7.24: Overload area maximum latency prediction curve and maximum latency we observe through our profiling for VPP single-core on the DUT 3

VPP is able to avoid the overload behavior to a certain degree. We show in Figure 7.24 how the maximum latencies behave in comparison to the number of ACL rules. In Figure 7.24, we observe that the maximum latency in the overload area does not change for 0, 50, and 100 ACL rules. This shows that VPP can delay the overload. Our model does not expect the latency to remain constant despite the firewall rule change. This leads to a situation where our model would predict negative latency by curve fitting up to 50 firewall rules. But this is not possible, so instead of negative latency, we predict no latency offset. Above 100 ACL rules we observe the linear increase of the latency. This shows that the VPP firewall, once it reaches the overload area, behaves similarly to iptables.

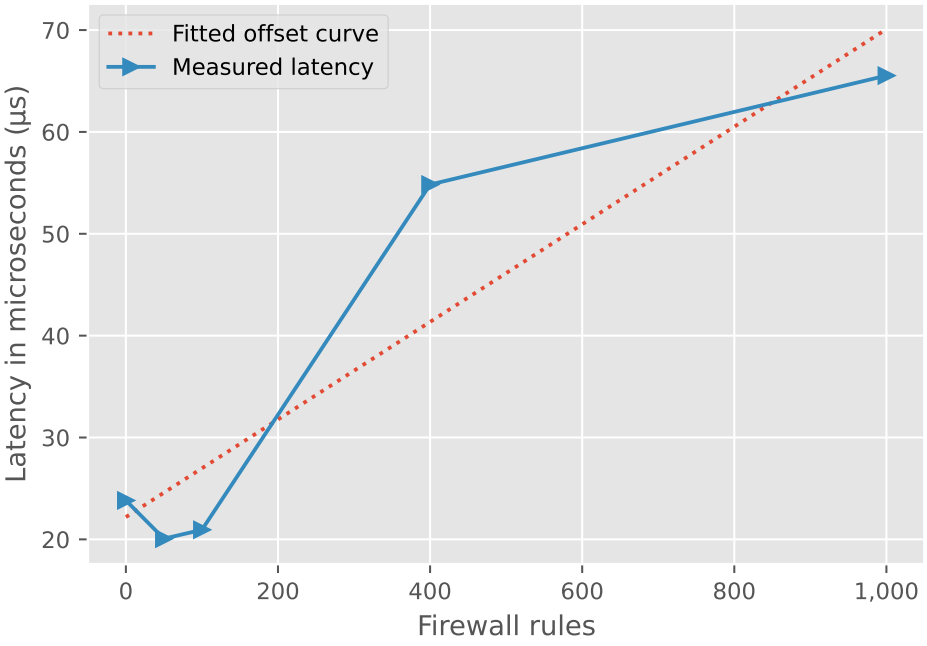


Figure 7.25: Non-overload area latency prediction curve and latency we observe through our profiling for VPP single-core on the DUT 3

For the non-overload area of the VPP firewall, we obtain a similar picture. In Figure 7.25, we show the latency curve of the non-overload range of the VPP firewall. The latency changes only slightly up to 100 ACL rules, after this point the latency increases more significantly.

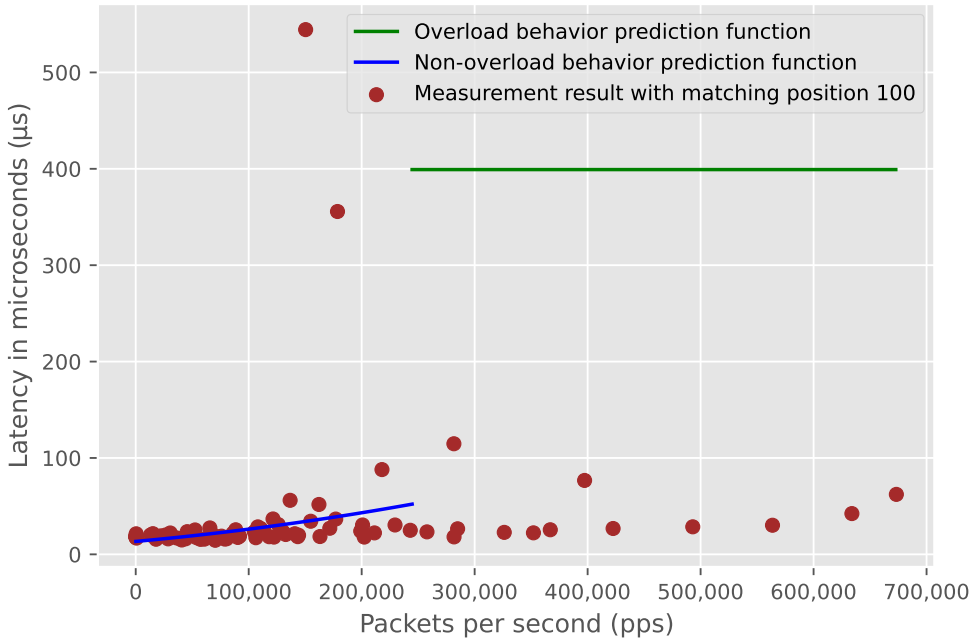


Figure 7.26: Comparison between VPP single-core latency behavior with 100 ACL rules and our VPP model latency behavior prediction for 100 ACL rules; The brown points originate from our profiling measurement with 100 firewall rules

In Figure 7.26, we show the behavior of the VPP firewall in comparison to our model with 100 ACL rules. We observe that our model predicts an overload area, but in reality, there is none. VPP can delay the overload area with its vectorized packet processing. Our maximum latency prediction function in Figure 7.24 and threshold prediction function obtain a deviation from the values we measured due to the non-linearity of the overload behavior of VPP. This deviation results in an prediction of an overload area below 100 ACL rules and a corresponding latency offset for the overload area. VPP can process up to 100 ACL rules traffic with almost no packet loss, thus it keeps the latency low. For the modeling in Figure 7.26, we achieve a mean deviation of $-51.96\ \mu s$. This means that we tend to over-predict the latency because our model is more conservative. The standard deviation for this measurement is $154.89\ \mu s$. The maximum deviation occurs at about $150,000$ pps and is $510.24\ \mu s$. The high standard deviation results from the wrong assumption that the VPP firewall has an overload behavior already at 100 ACL rules. In the overload are in Figure 7.26, where most large deviations occur, our model is too pessimistic. The latency over-prediction would lead to performance losses compared to the values we measured. However, this is not as bad as packets arriving too late, this happens when users work with a latency prediction that is too low, but packets actually experience significantly higher latency.

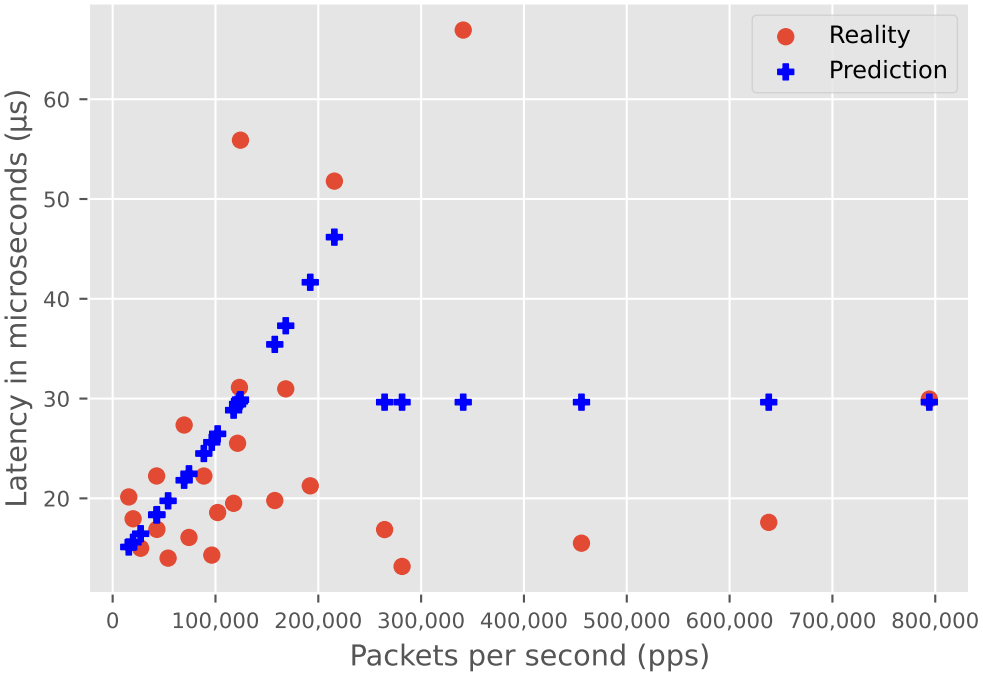
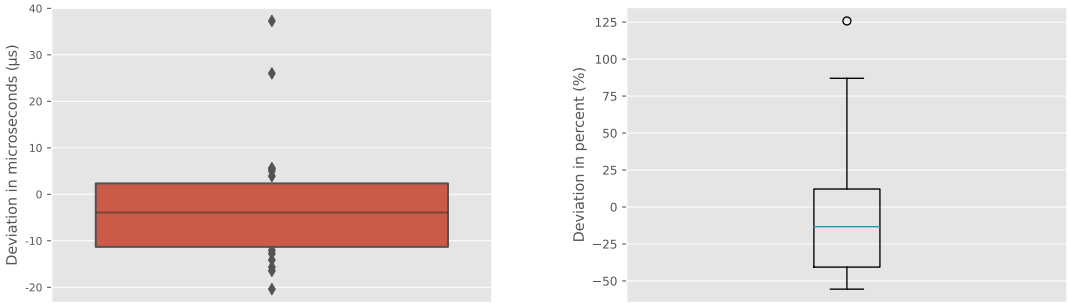


Figure 7.27: Comparison between VPP single-core latency measurement results and our model prediction functions based on our verification measurement; Latency in microseconds; The red points originate from our verification measurement without firewall rules

To verify our results, we generate additional data with our verification measurement that we did not test in our profiling. We use the same network settings as input for our model and compare our prediction with the real measured values. In Figure 7.27, we compare the two latency results. In the range below 200, 000 pps, the spread of the latencies is small and our model can predict the latencies with small deviations. With increasing spread, the deviation of our model becomes larger.

We illustrate the deviation between our prediction and our measured latency in Figure 7.28. Figure 7.28(a) shows the absolute deviation and Figure 7.28(b) shows the percentage deviation. The average latency deviation is -2.41 μs or -6.99 %. With an error probability of five percent, the true mean value lies within the confidence interval of -7.40 μs to 2.58 μs. The maximum deviation between our prediction and the real measured latency is 37.29 μs. In this case, at about 330, 000 pps, our model predicts the latency too low. The standard deviation in this case is 12.47 μs.



(a) Deviation in microseconds

(b) Deviation in percent

Figure 7.28: Deviation between our packet latency prediction with a firewall matching position at 1 and the actual packet latency of VPP single-core in microseconds and percent; We calculate the deviation from our prediction and the verification measurement without firewall rules

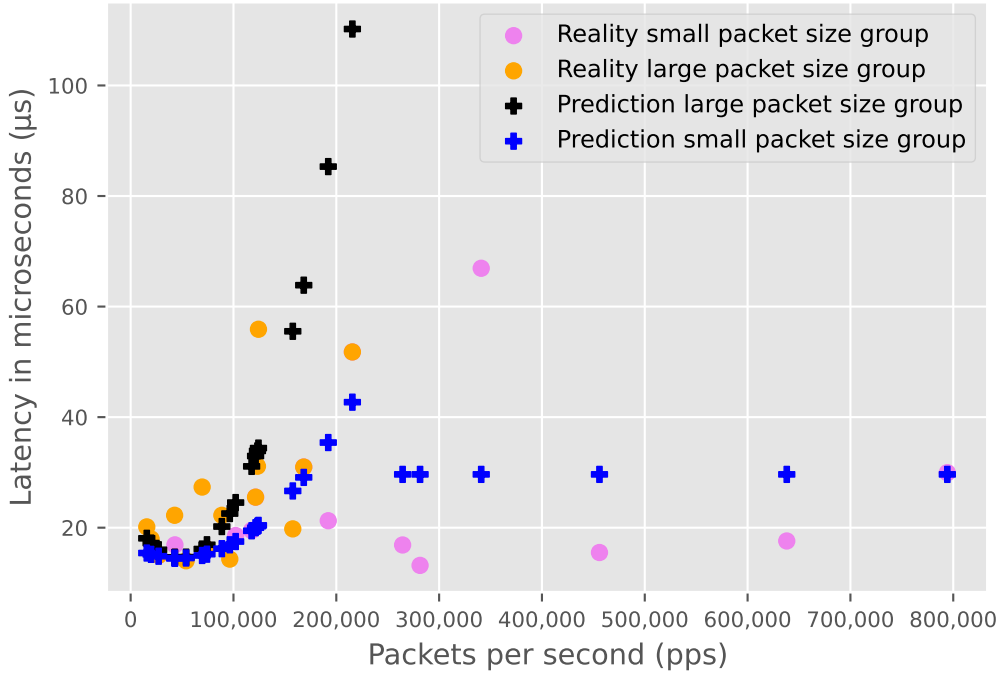
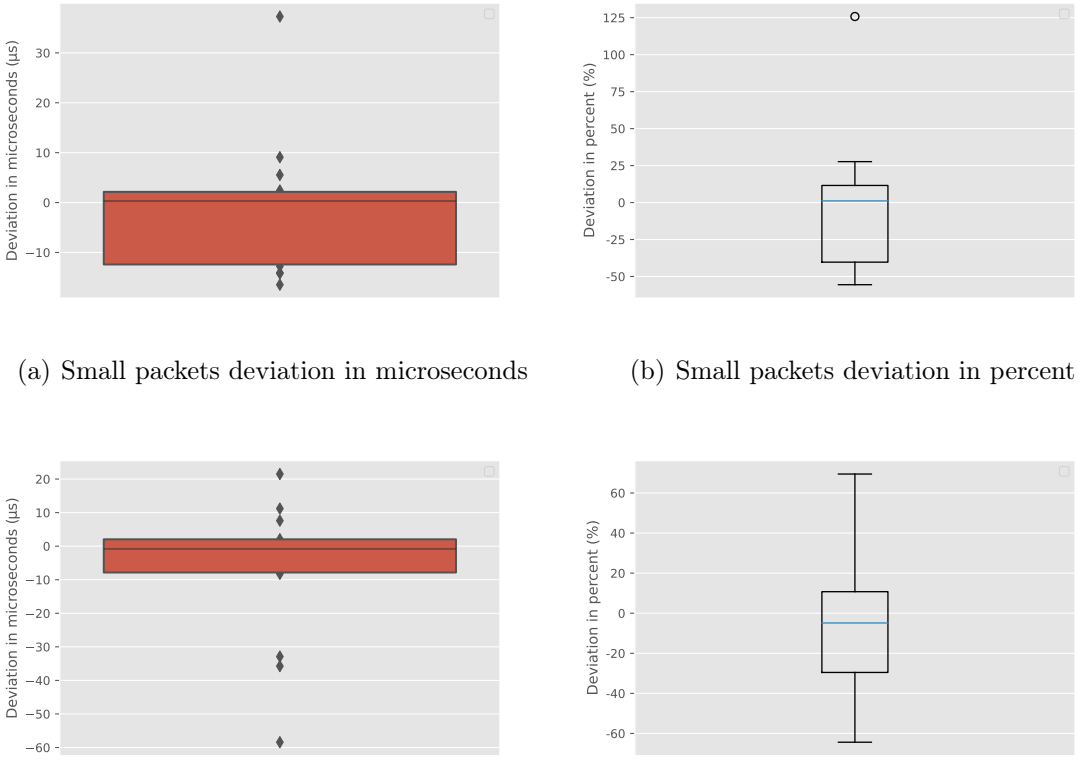


Figure 7.29: Comparison between VPP single-core without firewall rules configured: measurement results and our optimized model prediction functions based on our profiling; Latency in microseconds; The orange and violet points originate from our verification measurement without firewall rules

In Figure 7.29, we now consider the influence of our optimization on the prediction of the VPP single-core latency behavior. The average latency deviation for small packets with our optimization is -0.72 μs or -0.97 %. With our optimization, the average latency deviation for large packets is -6.85 μs or -2.81 %. Compared to Figure 7.27, we reduced the deviations. We show the deviations for the individual packet size groups in Figure 7.30. The standard deviation for small packets is now 12.87 μs and 19.75 μs for large packets. The maximum deviation is 58.41 μs. Through our optimization, we can predict the VPP with single-core settings even more accurate. This optimization helps us to make a more precise decision on the selection of VPP firewalls in industrial networks.



(a) Small packets deviation in microseconds

(b) Small packets deviation in percent

(c) Large packets deviation in microseconds

(d) Large packets deviation in percent

Figure 7.30: Deviation between our optimized packet latency prediction and the actual packet latency of the DUT 3 with single-core settings in microseconds and percent; We calculate the deviation from our prediction and the verification measurement without firewall rules; Top left and right represent the deviation for small packet sizes; Bottom left and right represent the deviation for large packet sizes

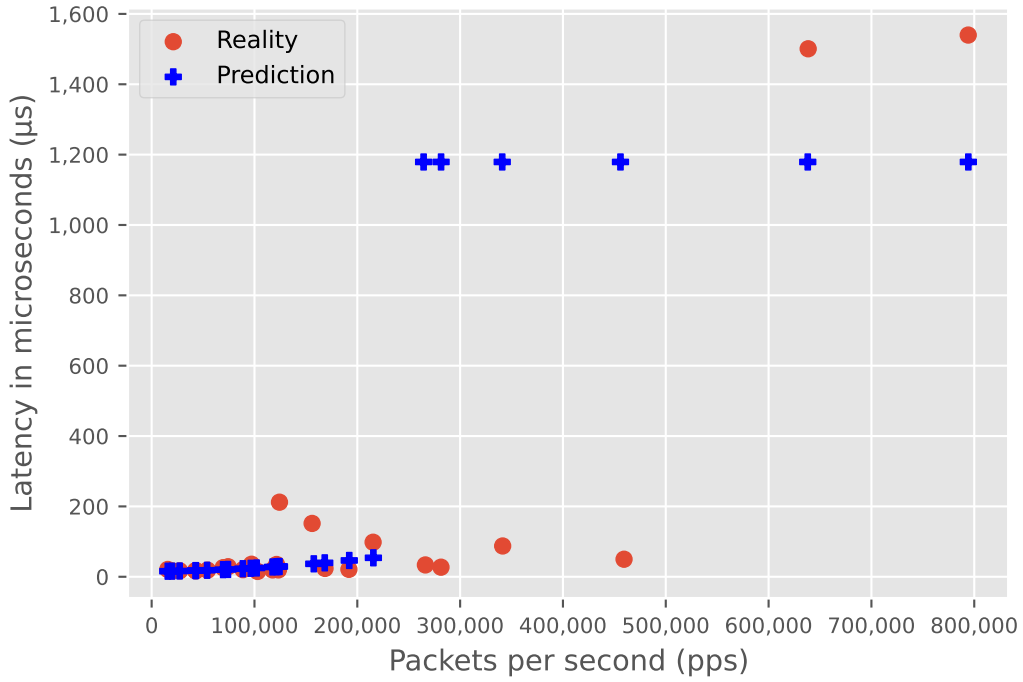
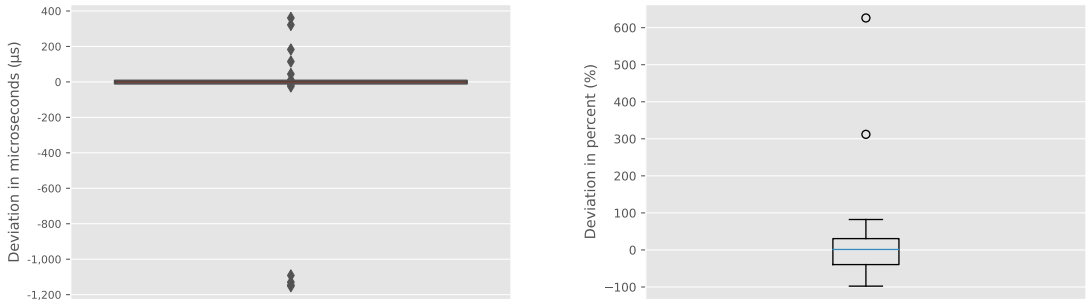


Figure 7.31: Comparison between VPP single-core with 200 firewall rules configured: measurement results and our model prediction functions based on our profiling; Latency in microseconds; The red points originate from our verification measurement with 200 firewall rules

In another verification measurement, we investigate how accurately our model predicts the behavior of the VPP firewall with the appropriate ACL rule at position 200. We compare the result in Figure 7.31. In this verification measurement we observe that the VPP firewall reaches the overload range at around 620,000 pps. Since our threshold prediction is based on the entire profiling history, our model cannot accurately determine the start of the overload behavior and tends to predict the threshold too early. In Figure 7.32, we present the deviation between our latency prediction and the latency we measured. The mean deviation of our prediction is $-141.16\ \mu s$ or 26.25 %. With an error probability of five percent, the true mean value lies within the confidence interval of $-318.13\ \mu s$ to $35.81\ \mu s$. Our model therefore tends to over-predict the latencies by $141.16\ \mu s$. The maximum deviation is $1151.92\ \mu s$. This is due to the incorrect threshold prediction. As a result, our model predicts some latencies significantly too high. The standard deviation is therefore $442.34\ \mu s$. However, the users would never want to achieve this overload in an industrial network, so these deviations are not relevant for the quality of our model. If we now consider only the non-overload area, our average deviation is $-183.10\ \mu s$ respectively 26.02 %. The early overload prediction leads to our model estimating the latencies too pessimistically.



(a) Deviation in microseconds

(b) Deviation in percent

Figure 7.32: Deviation between our packet latency prediction with a firewall matching position at 200 and the actual packet latency of VPP single-core in microseconds and percent; We calculate the deviation from our prediction and the verification measurement with 200 firewall rules

We show with our measurement results that the VPP firewall with single-core settings does not behave like an iptables firewall. With less than 100 ACL rules VPP can avoid its overload behavior. Above 100 ACL rules, the behavior becomes similar to that of an iptables firewall. Our model has difficulties in correctly predicting the behavior of the firewall, especially in the range below 100 ACL rules. This is due to the fact that our threshold prediction does not handle the constant behavior up to 100 ACL rules. Our model quantifies the industrial network capability of VPP and shows that VPP is able to meet the latency for cyclic traffic.

We show our prediction curve for the maximum packet loss of an VPP firewall with single-core setting in Figure 7.33. By fitting the root function to our measured packet loss, our function would predict a negative packet loss in the range below 50 ACL rules. This is not possible, so we predict 0 % packet loss for this range. In Figure 7.34, we compare the packet loss from our profiling and our packet loss prediction. We observe that packet loss occurs not as we expect it. On the one hand, the packet loss is very low at a maximum of 0.34 %, and on the other hand, the packet loss does not increase constantly. With this low packet loss, we can recognize every little outlier. The reason for this low packet loss is the way VPP processes the packets in single-core mode. In Figure 7.46, we show that VPP with multi-core settings generates less packet loss. Due to the processing of the packets on one CPU core only, there is a small amount of packet loss. Above 100 ACL rules, VPP reaches the overload range and thus the packet loss increases. Figure 7.33 illustrates that the packet loss above 100 ACL rules is similar to that of iptables firewalls.

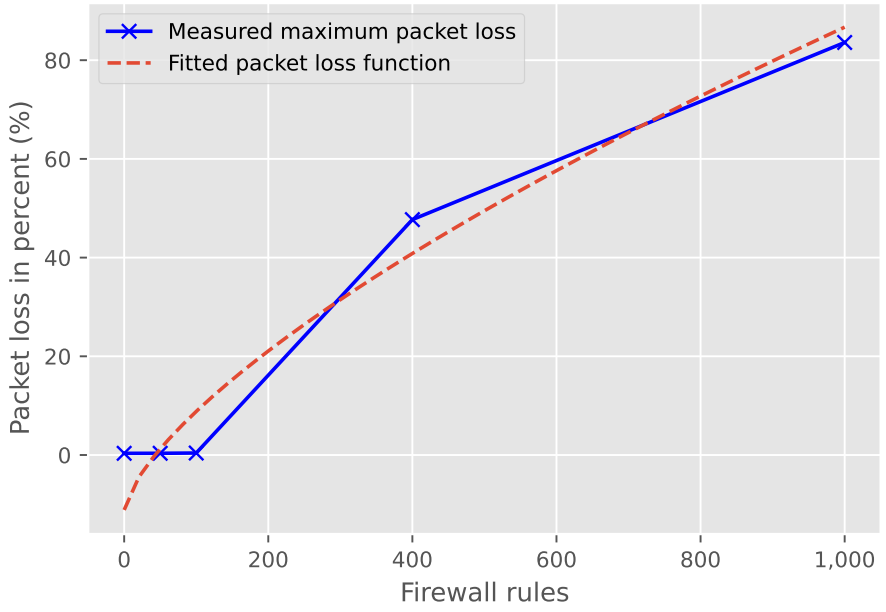


Figure 7.33: Packet loss function and measured packet loss from our profiling for VPP single-core on the DUT 3

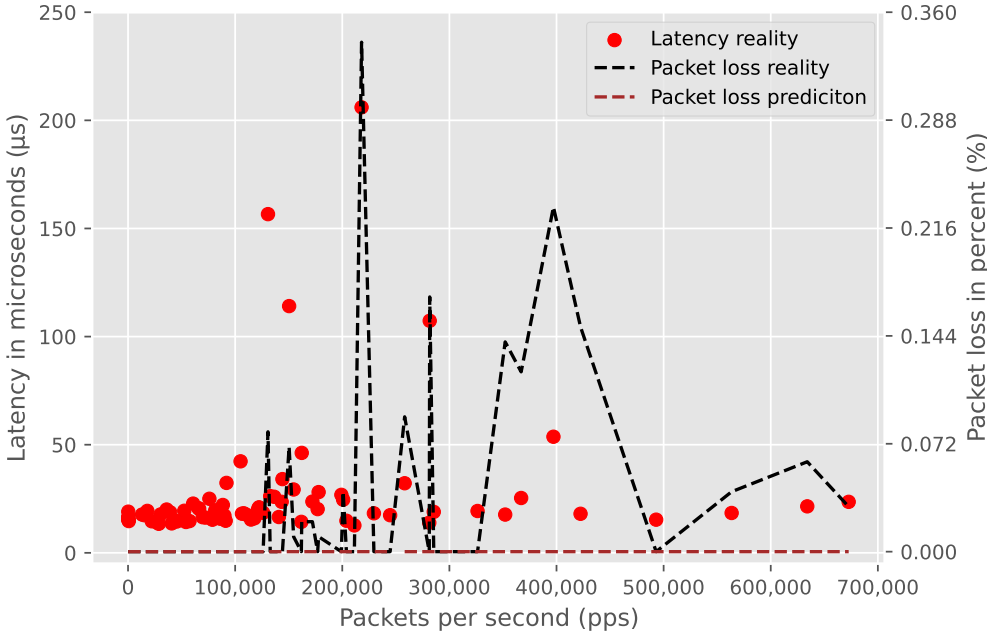


Figure 7.34: Comparison of our packet loss prediction and the packet loss we measured for VPP single-core on the DUT 3; The red points originate from our profiling measurement without firewall rules

7.1.3.2 VPP Multi-Core

VPP with multi-core settings enables the simultaneous processing of several packets on several CPU cores. With these settings, we profile the VPP firewall and use the results to create our model. In Figure 7.35, we show how our model represents the behavior of the VPP firewall. VPP does not show any noticeable overload behavior in comparison to the iptables firewalls. Due to the vectorized processing of up to 256 packets simultaneously, VPP prevents the overload area. The overall spread of the latencies across the entire measurement range is only 10.5 μs. Especially in the range below 100,000 pps, we measure a concentration of latency measurements at approximately 14 μs. Our model does not interpret the minor latency variations as overload behavior. Therefore, our model uses only the non-overload function for the latency prediction without firewall rules.

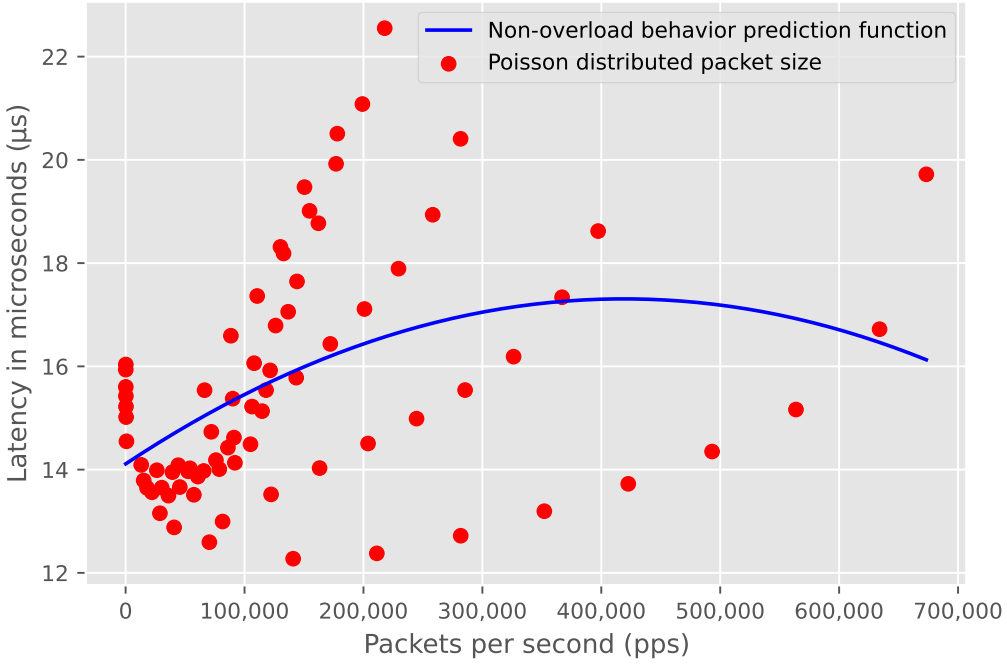


Figure 7.35: Comparison between VPP multi-core verification measurement results and our model prediction functions based on our profiling; The red points originate from our profiling measurement without firewall rules

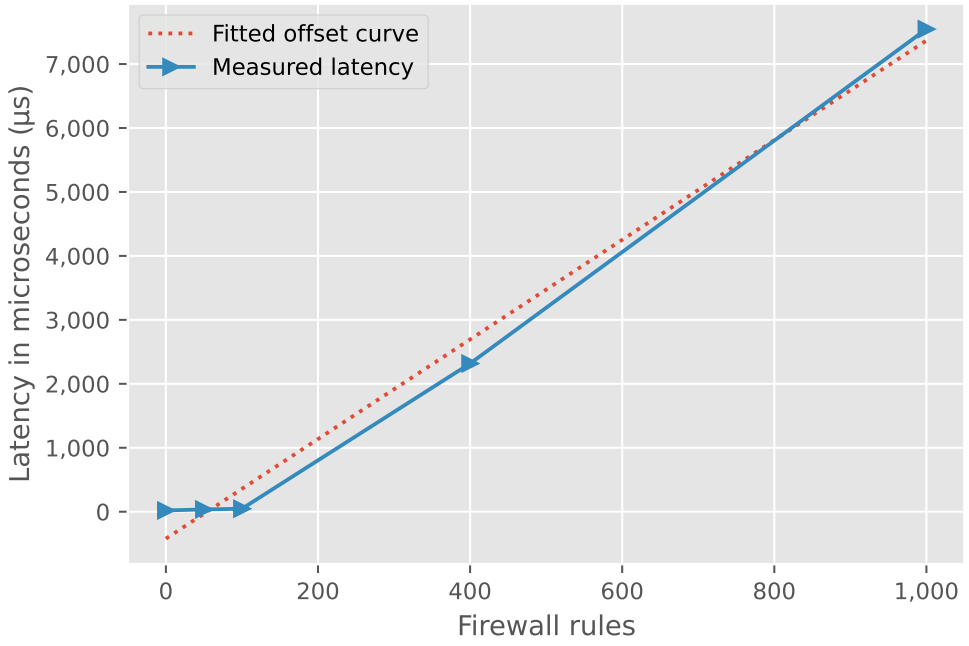


Figure 7.36: Overload area maximum latency prediction curve and latency we observe through our profiling for VPP multi-core on the DUT 3

The result for determining the latency in the overload range, in Figure 7.36 shows that the VPP firewall does not behave precisely the same way as iptables firewalls. The latency we measure on the VPP firewall stays nearly the same until we configure 100 ACL rules. As in the single core variant in Figure 7.24, our model would predict a negative latency offset up to 50 firewall rules. This is unrealistic, so our model does not predict a latency offset in this range. As we configure more than 100 ACL rules, the latency increases linearly like the iptables firewalls. Our model assumes a linear increase in latency. Hence, it cannot accurately represent the latency for less than 100 ACL rules.

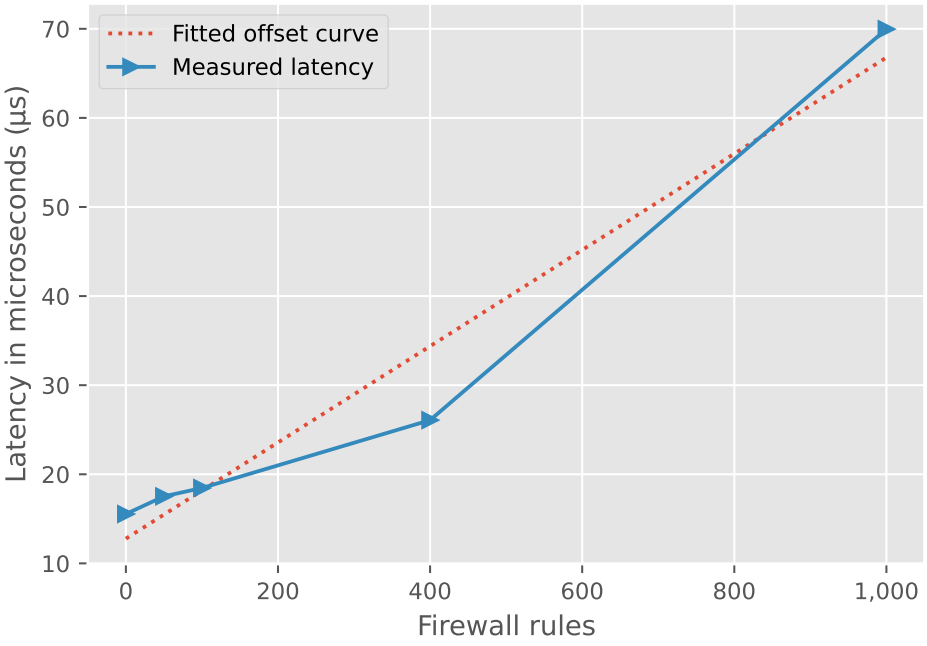


Figure 7.37: Non-overload area latency prediction curve and latency we observe through our profiling for VPP multi-core on the DUT 3

Since theVPP firewall has no overload area below 100 ACL rules, our model tries to map this. Figure 7.37 shows that the latency also increases in the non-overload area with increasing number of ACL rules. But just like the determination of the overload range, the latency hardly changes with less than 100 configured ACL rules.

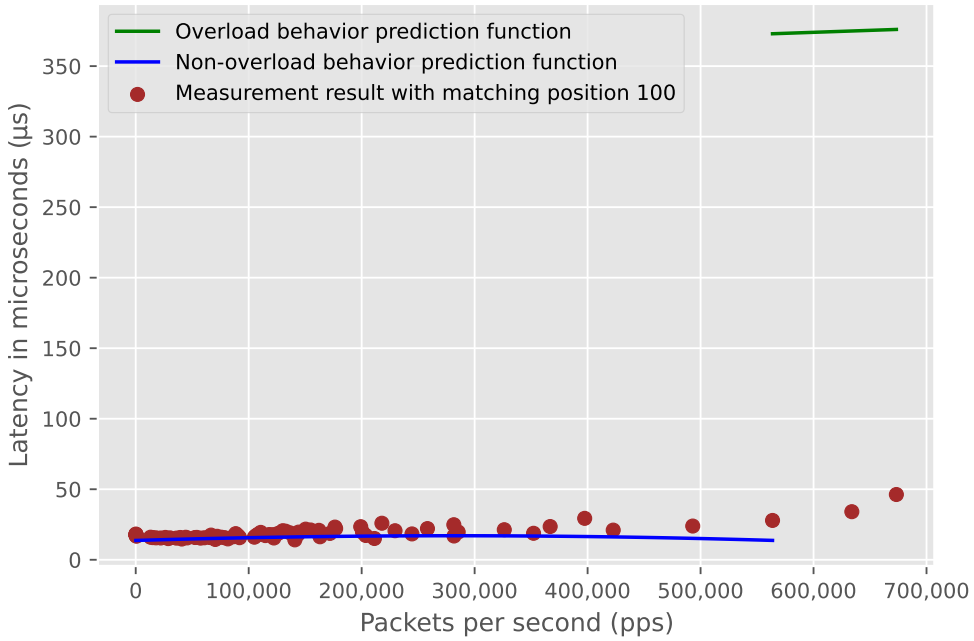


Figure 7.38: Comparison between VPP multi-core latency behavior with 100 ACL rules and our VPP model latency behavior prediction for 100 ACL rules; The brown points originate from our profiling measurement with 100 firewall rules

We show the latency prediction of our VPP model with 100 ACL rules configured in Figure 7.38. We observe that our model predicts an overload area, but in reality, the overload does not occur. The reason for the incorrect overload prediction is the same as in Figure 7.26. Our threshold prediction function predicts an increasing packet latency as a sign of overload behavior. This leads to the incorrect assumption that the VPP firewall is entering its overload area. Instead, the VPP firewall packet latency remains at less than 50 μs. The latency prediction of our model does not recognize that the VPP firewall has, in this case, no overload area. Therefore, our prediction in the range above $580,000$ pps is too high. In the range below $580,000$ pps, our model represents the behavior more realistically.

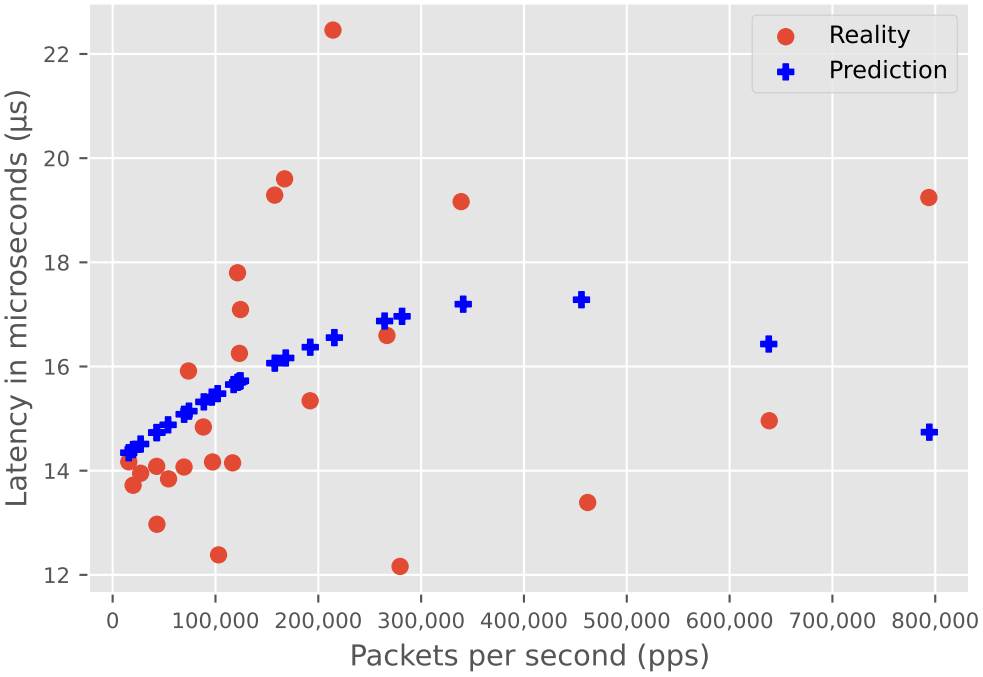
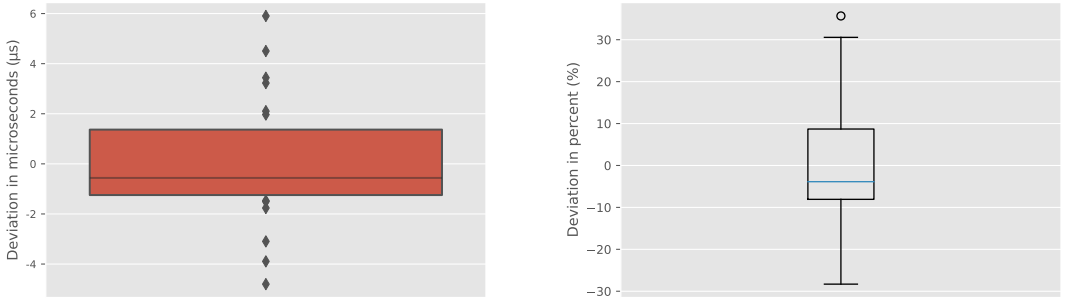


Figure 7.39: Comparison between VPP multi-core latency measurement results and our model prediction functions based on our verification measurement; Latency in microseconds; The red points originate from our verification measurement without firewall rules

We consider the deviations between the prediction and the real measurement in our verification measurement, in Figure 7.39. As for each model, we perform our verification measurement. We compare the packet loss that our model predicts with the packet loss we measure with the same input parameters. In Figure 7.39, we compare our latency prediction with the latencies we measure with our VPP firewall. In the range below 100, 000 pps, we can observe that the packet latencies of the VPP firewall have a variation of less than 4 μs. In this range, our model predicts the latencies closer to the actual latencies that we measure. Above 100, 000 pps, the latency variance we measure increases.



(a) Deviation in microseconds

(b) Deviation in percent

Figure 7.40: Deviation between our packet latency prediction and the actual packet latency of VPP multi-core in microseconds and percent; We calculate the deviation from our prediction and the verification measurement without firewall rules

Figure 7.40(a) shows the deviations between our latency measurement results and our latency prediction in microseconds. The average deviation in our verification test is 0.01 μs. With an error probability of five percent, the actual mean value is between -0.97 μs and 0.98 μs. On average, this means that our model predicts the latency 0.01 μs too low. The small deviation is because the VPP firewall generally generates very low packet latencies and has a low spread. If we look at the deviations on a percentage basis in Figure 7.40(b), we see that they are still lower than those of Figure 7.15(b). On average, our model for the VPP firewall predicts packet latencies to be 0.05 % lower than we measure. For the worst case in our verification measurement, our model predicts the latency to be 5.9 μs higher than we measure in reality. The standard deviation for our latency prediction is in this case 2.45 μs.

Due to the general low packet latency of the VPP firewall, the deviation between the latency prediction of our model and the latency we measure is quite small.

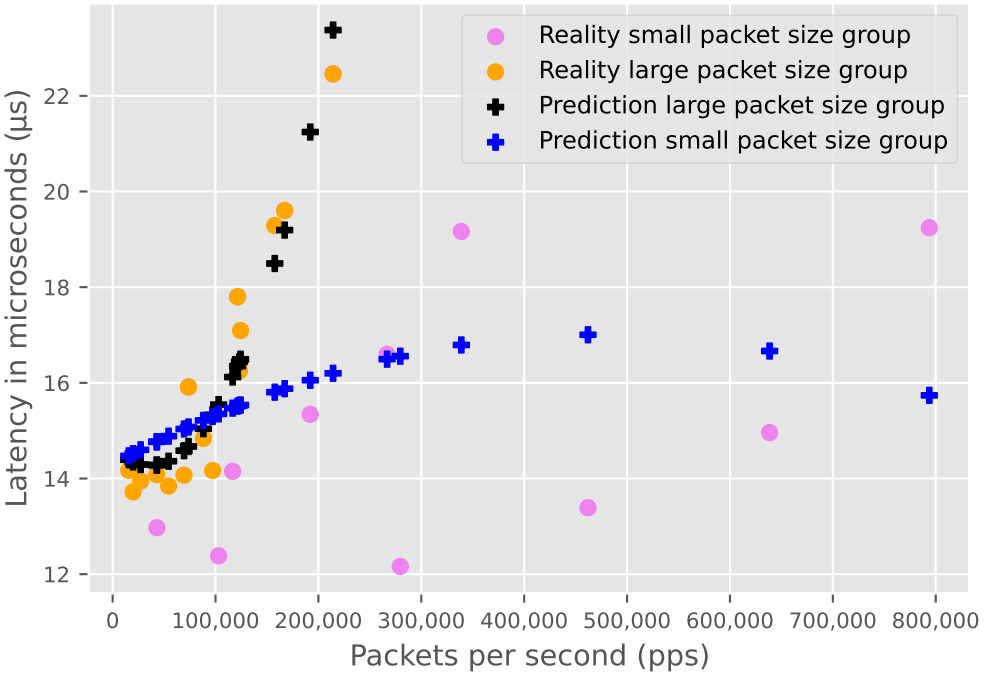
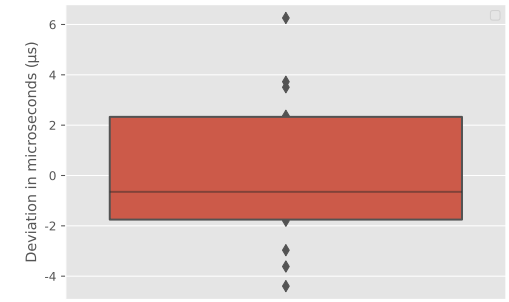
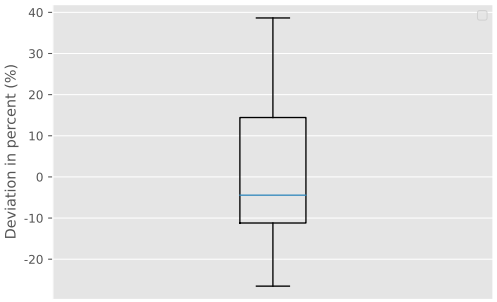


Figure 7.41: Comparison between VPP multi-core latency measurement results and our optimized model prediction functions based on our verification measurement; Latency in microseconds; The orange and violet points originate from our verification measurement without firewall rules

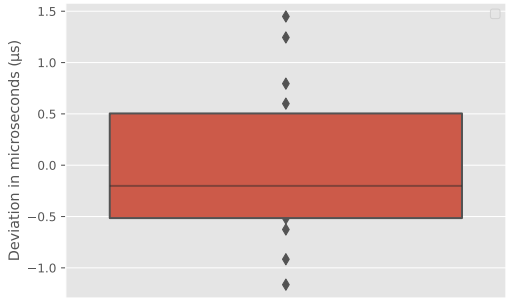
In our optimized prediction, we are now able to consider the packet sizes in a more differentiated way. In Figure 7.41, we compare the predictions for large and small packets with the latencies we measured without ACL rules. We note that the black pluses can also predict the higher packet latencies at $200,000$ pps, since they take into account the large packet sizes. The average latency deviation for small packets is 0.12 μs or 0.82 %. For large packets, we measure an average latency deviation of -0.02 μs or -0.22 %. Compared to Figure 7.39, we increase the accuracy of our model with our optimization. The optimization allows us to quantify the real-time capability more accurately.



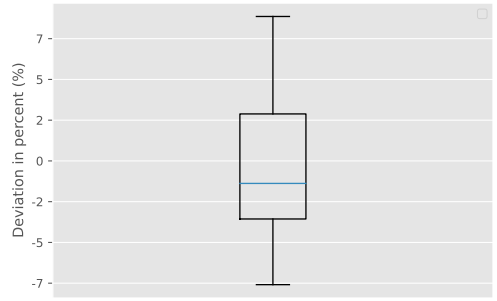
(a) Small packets deviation in microseconds



(b) Small packets deviation in percent



(c) Large packets deviation in microseconds



(d) Large packets deviation in percent

Figure 7.42: Deviation between our optimized packet latency prediction and the actual packet latency of the DUT 3 with multi-core settings in microseconds and percent; We calculate the deviation from our optimized prediction and the verification measurement without firewall rules; Top left and right represent the deviation for small packet sizes; Bottom left and right represent the deviation for large packet sizes

In Figure 7.42, we show the latency deviations for our optimized VPP multi-core prediction without firewall rules. We present the deviation for small packets in microseconds (Figure 7.42(a)) and in percent (Figure 7.42(b)). We illustrate the corresponding deviation for large packets in microseconds (Figure 7.42(c)) and in percent (Figure 7.42(d)). The maximum deviation for small packets is 6.25 μs and for large packets 1.44 μs. The standard deviation is 2.9 μs for small packets and 0.73 μs for large packets. We conclude that, in this case, we significantly improve the prediction for large packets and thus increase the overall prediction accuracy. VPP with multi-core settings meets in our verification measurement without firewall rules the latency, jitter, and packet loss requirements of cyclic traffic.

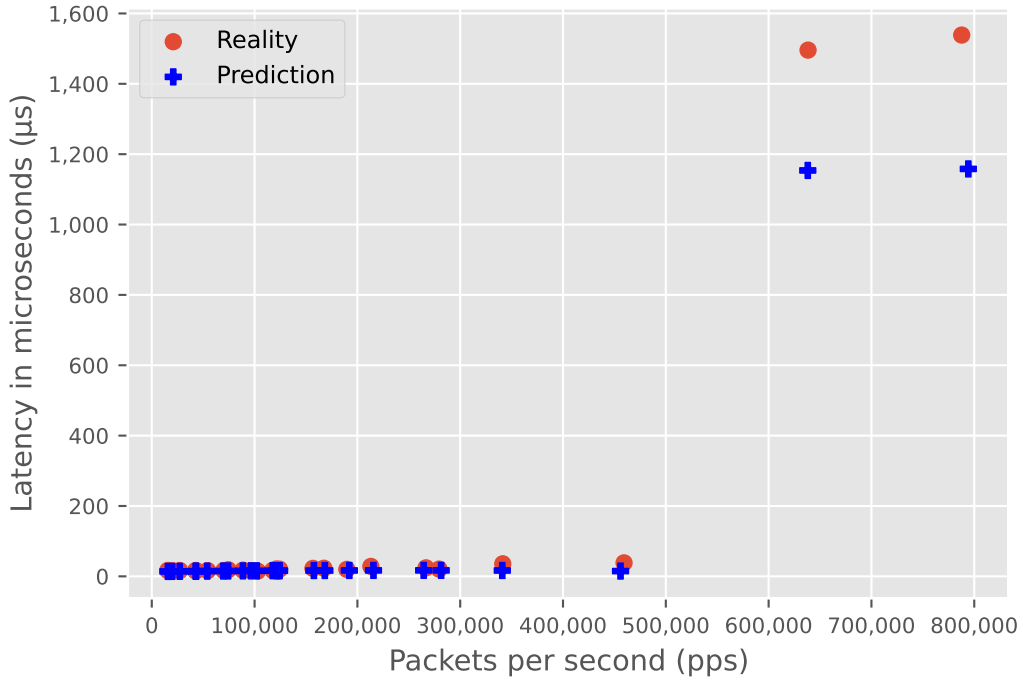


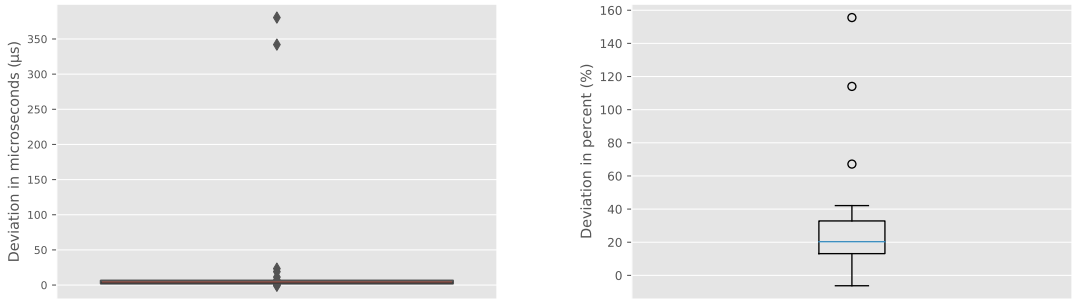
Figure 7.43: Comparison between VPP multi-core with 200 firewall rules configured: measurement results and our model prediction functions based on our profiling; Latency in microseconds; The red points originate from our verification measurement with 200 firewall rules

Our verification measurement with the matching firewall rule at position 200 produces the result shown in Figure 7.43. We observe that latency increases rapidly starting at about 620,000 pps. Since our model does not predict the overload threshold for VPP accurately, our model does not correctly determine these latencies. In an industrial network, we have to exclude the overload range, so the deviation of our model in this range has no relevance for the quality of our model.

In Figure 7.44, we show the absolut deviation (Figure 7.44(a)) and the percentual deviation (Figure 7.44(b)). We observe in Figure 7.44(a) that view predictions have a large deviation of more than 300 μs. Most of the deviations are around 0 μs. In Figure 7.44(b), we observe there are three outliers, although, in Figure 7.43, only two points are in the overload area. The three percentage outliers come from the points next to the threshold. This is because our curve fit function adjusts the quadratic function in the non overload area to have the smallest distances to all points. Therefore, these three percentage outliers result from the compromise of the curve fit function. Our mean deviation is 33.37 μs for the prediction in Figure 7.43. The average percentage deviation is thus 30.63 %. In this network configuration, our model tends to predict the latencies too low. The standard deviation is 96.98 μs. The largest deviation between our prediction and our measured latency is 380.47 μs. The high standard deviation is due to our predictions in the overload area. The largest

outlier in percentage terms occurs at approx. 460,000 pps. Our prediction in this case is 16 μs, but in reality we measure 38 μs.

If we now consider only the non-overload area, our average deviation is 4.8 μs and the standard deviation is 5.68 μs. This shows that our model predicts smaller deviations in the non-overload area. Our model is therefore able to quantify the behavior of the VPP firewall with multi-core settings in industrial networks.



(a) Deviation in microseconds

(b) Deviation in percent

Figure 7.44: Deviation between our packet latency prediction with a firewall matching position at 200 and the actual packet latency of VPP multi-core in microseconds and percent; We calculate the deviation from our prediction and the verification measurement with 200 firewall rules

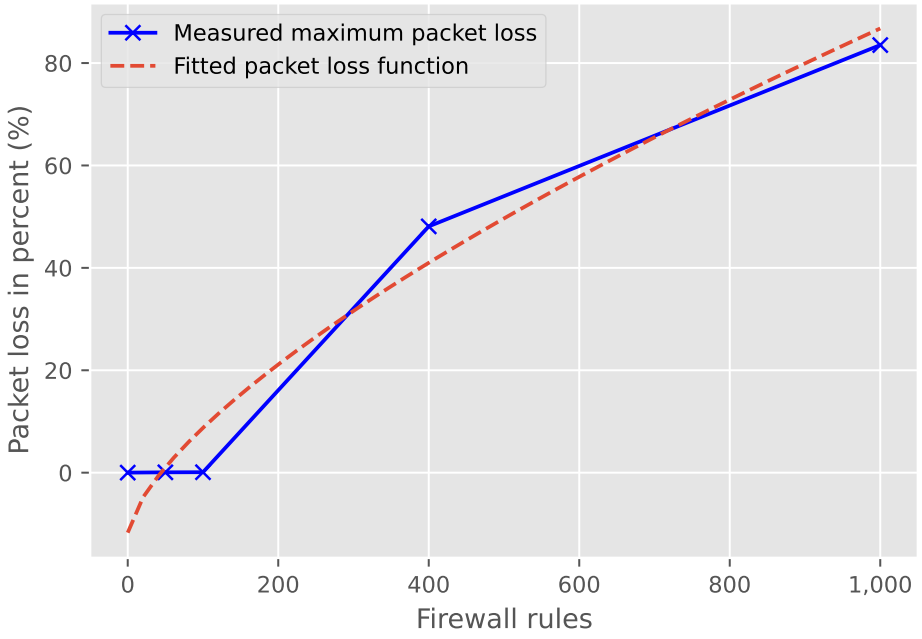


Figure 7.45: Packet loss function and measured packet loss from our profiling for VPP multi-core on the DUT 3

Part of our model is the prediction of packet loss. We show our prediction curve for the maximum packet loss of DUT 3 with multi-core setting in Figure 7.45. Below 100 ACL rules, we observe that the packet loss is ≈ 0 %. Above 100 ACL rules the packet loss rises til 83 %. VPP achieves the 0 % packet loss up til 100 ACL rules through vector processing. The vector processing delays packet loss and thus the overload area. By fitting the root function to our measured packet loss, our function would predict a negative packet loss in the range below 50 ACL rules. This is not possible, so we predict 0 % packet loss for this range. Above 100 ACL rules, VPP reaches the overload range, and thus the packet loss increases. Figure 7.45 illustrates that the packet loss above 100 ACL rules is similar to that of iptables firewalls.

Since the VPP firewall has no overload area below 100 ACL rules, we assume that packet loss must be low in this area. In Figure 7.46, we compare the packet loss from our profiling with 0 firewall rules (red points) and our packet loss prediction (brown dashed line) for this network traffic scenario. The black dashed line represents the actual packet loss we measure at 0 firewall rules. We observe that a packet loss of 0.01 % occurs for only one measurement point. The reason for low packet loss, in this case, is the measurement inaccuracy in our measurement setup. We determine packet loss by taking the difference between the packets transmitted and the packets received, there may be some inaccuracies. Packets, we generate at the end of the recording and do not receive before the end of the recording count as packet loss even though they are not lost from the DUT. Since VPP with multi-core settings uses several cores for traffic processing, there are no outliers in packet loss as with the single core settings.

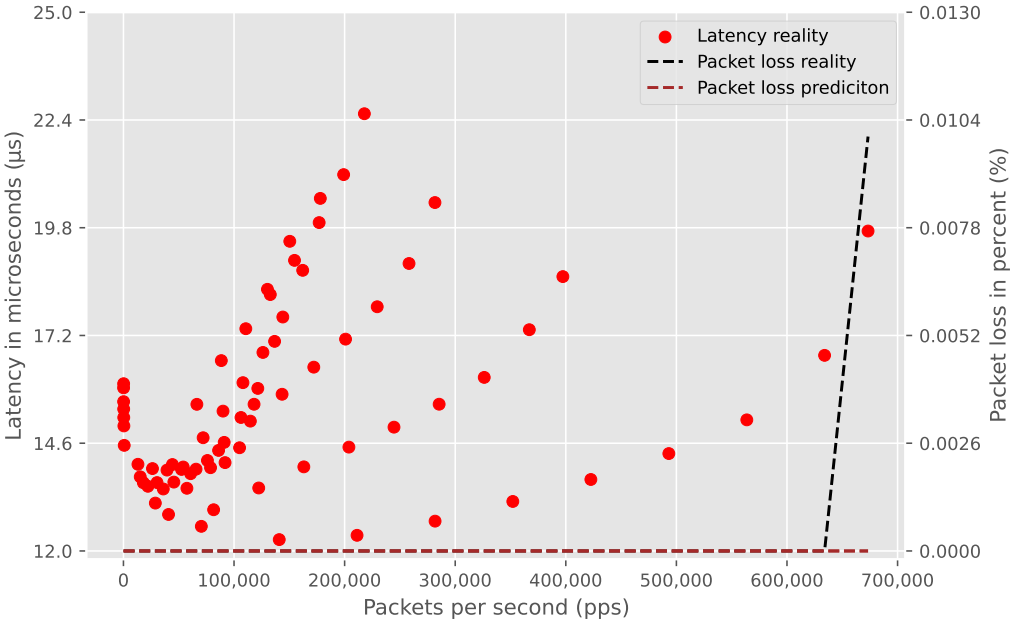


Figure 7.46: Comparison of our packet loss prediction and the packet loss we measured for VPP multi-core on our DUT 3; The red points originate from our profiling measurement without firewall rules

7.2 Applicability

Our model has different functions as we show in Chapter 6. Therefore, we use our model in different applications. In this section, we describe the different application scenarios and the benefits of our model. First, we discuss the applicability to different firewalls. We describe why our model can also predict the behavior of other firewalls. Second, we evaluate the duration of our profiling. Finally, we describe how our model can support the selection of network devices.

7.2.1 Applicability to other Firewalls

In Section 7.1, we demonstrate that the behavior of iptables is similar, and the behavior differs by the chosen hardware. Iptables implementations generally have similar behavior. The implementation details differ between kernel versions. Our model can predict the non-overload and overload behavior of iptables firewalls. The prediction works hardware independently. Our results show that our model is suitable for iptables firewalls. We discuss the results of DUT 1 in Section 7.1.1. For the DUT 1 without firewall rules, we can use our optimized model to accurately determine the latency of the small packets with a deviation of 5.18 % and the large packets with 7.52 %. With a maximum deviation of 22.88 μs in the behavior prediction of DUT 1, our model is precise enough for cyclic and acyclic traffic in TSN. Our model provides an accurate prediction to decide during the industrial network planning phase whether the DUT 1 is suitable or not.

In Section 7.1.2, we discuss our model prediction for the DUT 2 without firewall rules. Our optimized model predicts, on average, 91.36 μs or 55.31 % for small packets and 226.03 μs or 5.76 % for large packets higher latencies than we measure with the DUT 2. In an industrial network, packet loss is not allowed. If we predict too low latencies, packet loss can occur for time-critical data because the application does not wait forever to receive the data. Therefore, our latency predictions are more conservative. In addition, we recognize from our optimized model that even large packets cause overload in the DUT 2. The deviations with -91.36 μs and -226.03 μs are more significant than the deviations of the DUT 1. The reason for the higher deviation is that the latencies generated by the DUT 2 increase significantly with a slight increase in traffic. Between $8,000$ pps and $13,000$ pps, the latency increases from 200 μs to over $3,000$ μs. For the DUT 2, our prediction is not as accurate as for other DUTs but can also be used to assist in the network planning phase for device selection. Our deviations are still in a range that is sufficient for the prediction of cyclic TSN traffic up to 20 ms.

However, since the DUT 2 already has latencies of over 30 ms at 200 firewall rules, the DUT 2 is not sufficient for cyclic TSN traffic above $6,000$ pps. A more specific subdivision of the packet sizes is helpful to obtain a more accurate prediction for performance-weak devices. We did not take a closer look at an even more fine-grained subdivision of packet sizes in our optimization. Since the DUT 2 enters the overload area early, it can only be used in low-performance industrial networks.

In the case of VPP, our model is accurate in its prediction for both the single-core and multi-core variants, see Section 7.1.3. With single core settings, our prediction

is, on average, 0.72 μs or 0.97 % for small packets and 6.85 μs or 2.81 % for large packets above the measured latency. With multi-core settings, our model is even more accurate. The deviation here is 0.12 μs respectively 0.82 % for small packets and 0.02 μs respectively 0.22 % for large packets. We demonstrate that our model can predict the behavior of VPP firewalls accurately for industrial networks. The accuracy of our model is sufficient for cyclic and acyclic traffic in TSN.

We cannot test all possible software firewalls within the scope of this work. Thus, it is impossible to make a precise statement about the overall accuracy of our model. Apart from that, our profiling allows us to examine the behavior of a firewall with a few rules as well as with 1000 rules. Some firewalls, such as EAGLE30, allow a maximum of 2048 firewall rules. We can map the behavior of such iptables firewalls with our model as shown in Section 7.1. Further, we can give a general overview of the latency and packet loss behavior of software firewalls. This allows us to predict how a network device will behave at network creation time. A user can then make a decision, based on our prediction, about the use of firewalls in their industry network.

7.2.2 Profiling Duration

Another requirement for our model is the time it takes to create it. If the creation takes too long, the costs for generating the model will be too high. Thus, the benefit of the modeling will be smaller. A too short generation time can lead to the model becoming too inaccurate and therefore unusable. With our profiling, we have found a good trade-off for these problems. We describe the detailed profiling steps in Section 5.6. With our comparison between different firewall configurations in Section 6.2.1, we demonstrate that with our profiling, we have a deviation of 6.25 % between our customized threshold function and the actual thresholds. With more firewall configurations, our custom threshold function does not become more accurate. Our deviation from the measured thresholds increases to 7.66 % with four additional firewall rule configurations. In addition to the higher deviation, the profiling time increases by 80 % with the four additional firewall configurations. Our profiling offers the best compromise between profiling duration and accuracy.

7.2.3 Network Device Selection

In the previous Chapter 6 and Chapter 7, we describe how to construct our model and what results it delivers in terms of predicting the behavior of different software firewalls. In this section, we describe the applications where our model is useful.

Packet loss is not tolerated in industrial networks. With Profinet, for example, a maximum network load of 20 % with cyclic traffic is permitted. This is for safety reasons so that acyclic traffic does not lead to a loss of quality due to jitter. With our prediction, a lower utilization buffer can be used since our model can predict at what point a quality loss occurs due to packet loss. Our model can help, for example, with the selection of network devices. Within two hours, the profiling allows to build a model for the respective firewall. The model can predict the latency and packet loss for the respective firewall at given network and firewall configuration. These two factors are essential for the selection of industrial network

devices during network planning. There are additional requirements on the latency and jitter behavior besides the not allowed packet loss so that real-time critical traffic is possible. We demonstrate that with our optimized latency prediction, the latency can be predicted accurately enough for cyclic traffic. For example, if users have a specific latency requirement of 1 ms on their traffic, the model determines what firewall will meet this latency requirement up to a particular traffic volume. The user determines with our model the latency for a given pps value. We add the maximum deviation to our prediction. The result is the maximum latency the firewall is likely to have with this pps value. Based on the maximum latency, the user can now decide whether the firewall meets his requirements or not.

As we show in Figure 6.10, the jitter depends on the network situation and the cross traffic. Since the software firewalls are very jitter-prone, our model cannot predict their exact jitter behavior.

As we describe in Section 5.5.2, different applications have different tolerance levels to packet loss. Our model makes it possible to determine the behavior of the network devices before creating the network. Using our model, we can predict the behavior of the DUT under typical network traffic. By network typical traffic, we mean changing packet sizes and data rates. The results determine our network device selection. This way, we can match a selection of devices to the prevailing network traffic even before we set up the network. Our model thus saves money and valuable time during the setup of networks.

7.3 Model Limitations

There are countless different types of firewalls from various manufacturers. Different combinations of hardware and software firewalls are available. For example, there are Next-Generation Firewalls (NGFWs), application level gateways, stateful inspection firewalls, circuit level gateways, and packet filtering firewalls. We cannot examine all firewalls. Instead, we cover firewalls that allow stateful packet inspection and packet filtering. In our investigation of these firewalls, we were able to identify several points that are difficult to predict for our model. For example, interrupts on CPU cores can occur at any time. We cannot predict the occurrence of such interrupts. Nevertheless, they have a noticeable influence on the latency, as discussed in Section 6.1.7. Our model only considers the latencies and packet losses caused by interrupts during profiling. However, as we see in Figure 6.10(b), an interrupt can generate extreme worst-case latencies. Our model is not capable of accurately representing such a scenario. To adapt our model, we need more precise information about the frequency of the interrupts and their influence.

The connection tracking of VPP uses connection timeouts to avoid infinitely filling the hash table with connection entries. VPP deletes the corresponding entry in the hash table after a specific time if no packet from this connection occurs. Since this timeout can be set arbitrarily by the administrator, it is possible that the overhead of the connection tracking increases. The timeout can be set so small that for each packet of a connection, the additional connection entry overhead is necessary. Since our model does not know the exact timeouts, we cannot estimate the effects of this

edge case.

Points like these cause our model not to be able to predict every network device's behavior accurately.

7.4 Measurement Limitations

The network traffic that a firewall has to process can differ from network to network. In some networks, the data rate changes constantly, and packet sizes vary from small to large. In other networks, small packets are predominant, and the data rate is mostly constant. In our measurements, we cannot simulate every possible network traffic.

7.4.1 General Limitations

To create our model, we try to generate generally realistic network traffic. Therefore, we use the Poisson distribution for our packet size generation. We also change the data rate to observe the behavior of the DUT under varying conditions. With this method, we can form a general behavior pattern of the DUT. However, we do not profile specific types of network traffic with this method. After profiling, our model offers the option of specifying whether small or large packets are more likely to occur in the network. With this specification, our model then uses rather large or small packets of the profiling for its prediction.

7.4.2 Packet Loss Measurement Limitations

To determine the packet loss, we compare the number of packets we generate and the number of packets we analyze in our MoonGen script. The difference between them determines our packet loss. Due to our packet generator warm-up phase, we already generate packets for five seconds until our recording starts. However, the counter for the generated and analyzed packets already counts in the warm-up phase. We do so because some packets we generate before the recording start are part of the analysis taking place during the recording. Hence, we also consider these packets for the packet loss calculation. In addition, we count packets that we generate shortly before the end of the recording but do not process at the packet generator before the end of the recording. Packets that MoonGen generates during recording and never processes because the recording ends before occur only in combination with very small packets (64 bytes) and a high data rate (1000 MBit/s). At $700,000$ pps, this means ≈ 35 packets that MoonGen incorrectly measures as packet loss. Accordingly, this type of packet loss only occurs in edge cases. Without adding additional IDs in the packets, we cannot trace the actual packet loss. The additional overhead of checking each packet for packet loss, however, results in packet loss itself. With a small packet size and a high data rate, the generator cannot check all packets on the fly for packet loss. This leads to packet loss at the ingress of the packet generator.

7.5 Improvements for Iptables

In our research, we found that VPP offers lower latency and packet loss than iptables. Additionally, the behavior of stateful rules is different. In VPP, the latency behaves static for stateful rules, but in iptables with stateful rules, it behaves linearly. By examining the packet flow in each firewall, we can show why the stateful rule behaves differently in both firewalls. Despite the match in the connection table, iptables checks each filter rule until it finds the matching one. This leads to no performance optimization in comparison to stateless rules.

If VPP encounters a match in the connection table, VPP forwards the packet directly. Iptables still checks the filter rules after a match to ensure that there are no outdated entries in the connection table. To improve the stateful rules, iptables must clear the entire connection table after each rule change. As a result, iptables has the same advantages and disadvantages as VPP when dealing with stateful rules. With this optimization, iptables must create a new entry in the connection table for each new connection, even if it already existed before flushing the table. But the advantage of this optimization is that iptables can forward packets directly if there is a matching entry in the connection table. The stateful optimization means connections with connection tracking can communicate faster and tend to have less packet loss.

8 Conclusion

In this thesis, we created and simulated a model that predicts the latency, jitter, and pack loss behavior of firewalls on network performance. In the following, we first summarize the results and findings of this thesis. Afterward, we provide an outlook for future work.

8.1 Summary

In Chapter 5, we defined the framework conditions for our model and the measurement environment. To generate our network traffic most realistically, we varied the packet size and data rate. We exposed the DUT to realistic network scenarios to provide valuable results.

On the basis of the system model in Chapter 6, we determine and investigate the parameters influencing the network performance. The parameters that influence the behavior of a firewall are the network traffic, the firewall rules, and cross traffic that creates additional interrupts on the firewall. We combine the findings from these investigations into our prediction model. The base version of our model can predict the latency of the firewall without firewall rule configurations. For industrial applications, it is essential to know the amount of network-related packet loss to ensure reliable operation. Therefore, we also included packet loss prediction in our model. Using the insights gained from our firewall rule configurations investigations, we were able to adjust our model such that it predicts the behavior of the firewall with firewall rules. Once more detailed information on network traffic packet sizes is available, our model provides the ability to include this information in the prediction and thus predict more accurate results. Our investigations of network traffic revealed that the change in data rate has a direct influence on the latency behavior of the firewall. Therefore, we allow our model to make predictions in the time interval. We do so by specifying the interval and the parameters that change over time. Our model can use this information to predict the latency curve over time. To automate the creation of models for new firewalls, we developed a profiling script. The profiling script tests different firewall rule configurations, measures the latency and packet loss behavior of a firewall, and makes the results available for modeling.

In Chapter 7, we demonstrate the accuracy of our model by applying it to other firewalls. Therefore, we present the behavior predictions of our model for different software firewalls. We conclude (with our results) that iptables firewalls behave similarly. The higher latency and jitter of DUT 2 in comparison to DUT 1 is due to the difference in hardware performance. We define the term overload area for software firewalls as the area where packet loss in combination with high latency occurs. We show that the VPP firewall delays the overload area as much as possible through its vector-based processing. In the overload area, the latency of VPP

behaves like that of an iptables firewall. For sending real-time critical data across network boundaries, information about the behavior of the firewalls at the edge of the networks is necessary. The user can select the appropriate firewalls based on the information regarding latency, jitter, and packet loss. We demonstrate with our results that our model is suitable to support the planning phase of industrial networks. Our model predicts firewall latency performance accurately enough for cyclic and acyclic TSN traffic to enable device selection decisions based on these results. The firewall behavior prediction makes it possible to decide during the network planning phase which firewalls are capable of serving the required quality of service.

8.2 Future Work

With our model, we predict the performance impact of network traffic on software firewalls. The following section first describes the ideas we still have regarding our model and profiling. Finally, we describe the simulation possibilities we would like to explore with our model.

8.2.1 Open Source Profiling

The further development of our model for more precise predictions requires more data from various firewalls. It is useful to publish our profiling to get as much data about different firewalls as possible. To publish our profiling of firewalls, it is essential that the time stamping also works without an extra time stamping switch. Therefore, we have to investigate how applicable the time stamping function of the MoonGen packet generator is. Not only can the performance of the packet generator be affected, but the precision of the time stamps can also vary due to the additional overhead. This behavior only occurs when MoonGen generates and processes small packets (64 Bytes) with a high data rate (1000 MBit/s). This combination corresponds to a boundary condition and is not decisive for the overall network traffic. When evaluating the time stamps, we found that the performance of our packet generator suffers from too many operations during runtime. The hardware on which our package generator runs limits its performance. Depending on how huge the influence on the time stamping is, it will affect the prediction of the corner case. For iptables firewalls, this boundary condition affects the overload area. Firewalls in industrial networks must not operate in the overload area to avoid packet loss. Since we design our model for the consideration of firewalls in industrial networks, the overload area does not matter. Nevertheless, we must investigate and exclude such a measurement error prior to publication.

8.2.2 Simulation with OMNeT++

Other research uses simulation tools to show the applicability of scheduling and quality of service. For instance, the Network Simulator for Time-Sensitive Networking (NeSTiNg) project uses *OMNeT++* to simulate TSN [50]. Simulation frameworks such as *OMNeT++* save time and money because there is no need for us to build

up the network structure in reality. So far, there is no firewall behavior model like ours available for a simulation tool. A simulation framework offers the possibility to model networks. We can include our model in this network and test what behavior it predicts. Therefore, it is necessary to integrate our model into a simulation framework such as *OMNeT++* to simulate the prediction of our model in different networks.

Bibliography

[1] *Questions and Answers about TSN*, Siemens, Jul. 2021, p. 9. [Online]. Available: https://cache.industry.siemens.com/dl/files/263/109757263/att_1075932/v1/109757263_TSN_QA_V2.0_en.pdf (visited on 06/12/2022).

[2] L. Wüsteney, M. Menth, R. Hummen, and T. Heer, "Impact of Packet Filtering on Time-Sensitive Networking Traffic," 2021. DOI: 10.1109/WFCS46889.2021.9483611.

[3] *Was ist Industrie 4.0?* Bundesministerium für Wirtschaft und Klimaschutz. [Online]. Available: https://www.plattform-i40.de/IP/Navigation/DE/Industrie40/WasIndustrie40/was-ist-industrie-40.html (visited on 06/12/2022).

[4] J. M. Stewart, *Network Security, Firewalls and VPNs*, ser. Jones & Bartlett Learning information systems security & assurance series. Jones & Bartlett Learning, 2013, ISBN: 9781284031683. [Online]. Available: https://books.google.de/books?id=qZgtAAAAQBAJ.

[5] W. Noonan and I. Dubrawsky, *Firewall Fundamentals*. Cisco Press, 2006, ISBN: 1-58705-221-0.

[6] J. M. Kizza, *Guide to Computer Network Security*. Springer Cham, 2020, ISBN: 978-3-030-38141-7.

[7] *Netfilter Conntrack Sysfs variables*. [Online]. Available: https://docs.kernel.org/networking/nf_conntrack-sysctl.html (visited on 06/05/2022).

[8] A. L. DeCarlo and R. G. Ferrell. (Jan. 1, 2021). The 5 different types of firewalls explained, [Online]. Available: https://www.techtarget.com/searchsecurity/feature/The-five-different-types-of-firewalls (visited on 06/09/2022).

[9] D. G. Güttich. (Oct. 6, 2017). Grundlagen der Next Generation Firewall (NGFW), [Online]. Available: https://www.security-insider.de/grundlagen-der-next-generation-firewall-ngfw-a-648098/ (visited on 06/09/2022).

[10] T. Bradley. (Oct. 24, 2021). What Is a Firewall and How Does a Firewall Work? [Online]. Available: https://www.lifewire.com/what-is-a-firewall-2487290 (visited on 06/09/2022).

[11] (Mar. 5, 2020). Networking, [Online]. Available: https://linux-kernel-labs.github.io/refs/heads/master/labs/networking.html (visited on 06/09/2022).

[12] J. Ellingwood. (Aug. 20, 2015). A Deep Dive into Iptables and Netfilter Architecture, [Online]. Available: https://www.digitalocean.com/community/tutorials/a-deep-dive-into-iptables-and-netfilter-architecture (visited on 06/09/2022).

[13] R. Russell and M. Neuling, *iptables(8) - Linux man page*. [Online]. Available: https://linux.die.net/man/8/iptables (visited on 06/09/2022).

[14] A. Stender. (May 2022). Connection tracking (conntrack) - Part 1: Modules and Hooks, [Online]. Available: https://thermalcircle.de/doku.php?id=blog:linux:connection_tracking_1_modules_and_hooks (visited on 05/25/2022).

[15] R. Rosen. (Apr. 23, 2018). Userspace Networking with DPDK, [Online]. Available: https://www.linuxjournal.com/content/userspace-networking-dpdk#:~:text=It's%20a%20multi%2Dvendor%20and,Linux%20Foundation%20in%20April%202017. (visited on 06/10/2022).

[16] M. Tatarski, *Effiziente und schnelle Verarbeitung von Netzwerkpakten mittels DPDK*. Universität Rostock, 2021. [Online]. Available: https://doi.org/10.18453/rosdok_id00003057 (visited on 06/10/2022).

[17] (Jun. 9, 2022). What is the Vector Packet Processor (VPP), [Online]. Available: https://s3-docs.fd.io/vpp/22.06/ (visited on 06/10/2022).

[18] A. Toonk. (Apr. 5, 2020). Kernel Bypass Networking With FD.io and VPP., [Online]. Available: https://medium.com/swlh/kernel-bypass-networking-with-fd-io-and-vpp-fc3a53a669f9 (visited on 06/10/2022).

[19] (2022). Scalar vs Vector packet processing, [Online]. Available: https://s3-docs.fd.io/vpp/22.06/aboutvpp/scalar-vs-vector-packet-processing.html (visited on 06/10/2022).

[20] P. Barry and P. Crowley, "Chapter 5 - Embedded Processor Architecture," in *Modern Embedded Computing*, P. Barry and P. Crowley, Eds., Boston: Morgan Kaufmann, 2012, p. 150, ISBN: 978-0-12-391490-3. [Online]. Available: https://learning.oreilly.com/library/view/modern-embedded-computing/9780123914903/xhtml/CHP005.html (visited on 06/10/2022).

[21] *PC Cooling: The Importance of Keeping Your PC Cool*. [Online]. Available: https://www.intel.com/content/www/us/en/gaming/resources/pc-cooling-the-importance-of-keeping-your-pc-cool.html (visited on 06/10/2022).

[22] A. Salim, "The Effect of Temperature on the Electrical Conductor," May 1, 2018. [Online]. Available: https://www.ijecce.org/administrator/components/com_jresearch/files/publications/IJECCE_4227_FINAL.pdf (visited on 06/10/2022).

[23] H. Hofmann, K. Kafadar, and H. Wickham. (Dec. 2, 2011). Letter-value plots: Boxplots for large data, [Online]. Available: https://vita.had.co.nz/papers/letter-value-plot.pdf (visited on 06/30/2022).

[24] H. Schmid, *Elementare Technomathematik*. Springer Spektrum Berlin, Heidelberg, 2018, p. 229, ISBN: 978-3-662-58008-0. DOI: https://doi.org/10.1007/978-3-662-58008-0. [Online]. Available: https://link.springer.com/content/pdf/10.1007/978-3-662-58008-0.pdf (visited on 06/10/2022).

[25] J. Brownlee. (Nov. 14, 2021). Curve Fitting With Python, [Online]. Available: https://machinelearningmastery.com/curve-fitting-with-python/#:~:text=Curve%20fitting%20is%20a%20type,examples%20of%20inputs%20to%20outputs. (visited on 06/10/2022).

[26] *What is OMNeT++?* [Online]. Available: https://omnetpp.org/intro/ (visited on 06/11/2022).

[27] A. Varga, *OMNeT++, Discrete Event Simulation System*, Jun. 15, 2003. [Online]. Available: http://src.gnu-darwin.org/ports/science/omnetpp/work/omnetpp-2.3p1/doc/usman.pdf (visited on 06/11/2022).

[28] N. I. of Standards and Technology, "Guide to Industrial Control Systems (ICS) Security," U.S. Department of Commerce, Washington, D.C., Tech. Rep. National Institute of Standards and Technology Special Publication 800-82, Revision 2, 2015. DOI: 10.6028/NIST.SP.800-82r2.

[29] R. Belliardi, J. Dorr, T. Enzinger, F. Essler, J. Farkas, M. Hantel, M. Riegel, M.-P. Stanica, G. Steindl, R. Wamßer, K. Weber, and S. Zuponcic. (Sep. 13, 2018). Use Cases IEC/IEEE 60802, [Online]. Available: https://www.ieee802.org/1/files/public/docs2018/60802-industrial-use-cases-0918-v13.pdf (visited on 06/29/2022).

[30] E. D. Knapp and J. Langill, *Industrial Network Security: Securing Critical Infrastructure Networks for Smart Grid, SCADA, and Other Industrial Control Systems*. Elsevier Science, 2014, ISBN: 9780124201842. [Online]. Available: https://learning.oreilly.com/library/view/industrial-network-security/9780124201149/B9780124201149000095/B9780124201149000095.xhtml#st9010.

[31] T. McGuiness, "Defense In Depth," Nov. 11, 2001. [Online]. Available: https://sansorg.egnyte.com/dl/B3uguj8Rvo (visited on 06/11/2022).

[32] B. Hickman, T. Martin, S. Tadjudin, and D. Newman, *Benchmarking Methodology for Firewall Performance*, RFC 3511, Apr. 2003. DOI: 10.17487/RFC3511. [Online]. Available: https://rfc-editor.org/rfc/rfc3511.txt.

[33] B. Balarajah, C. Rossenhoevel, and B. Monkman, "Benchmarking Methodology for Network Security Device Performance," Internet Engineering Task Force, Internet-Draft draft-ietf-bmwg-ngfw-performance-13, Jan. 2022, visited on 21-01-2022, 59 pp. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-bmwg-ngfw-performance-13.

[34] D. Zvabva, P. Zavarsky, S. Butakov, and J. Luswata, "Evaluation of Industrial Firewall Performance Issues in Automation and Control Networks," *29th Biennial Symposium on Communications, BSC 2018*, DOI: 10.1109/BSC.2018.8494696.

[35] M. Cheminod, L. Durante, A. Valenzano, and C. Zunino, "Performance impact of commercial industrial firewalls on networked control systems," *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, 2016, ISSN: 19460759. DOI: 10.1109/ETFA.2016.7733576.

[36] M. Cheminod, L. Durante, L. Seno, and A. Valenzano, "Performance Evaluation and Modeling of an Industrial Application-Layer Firewall," *IEEE Transactions on Industrial Informatics*, pp. 2159–2170, 2018, ISSN: 15513203. DOI: 10.1109/TII.2018.2802903.

[37] D. Scholz, H. Harkous, S. Gallenmüller, H. Stubbe, M. Helm, B. Jaeger, N. Deric, E. Goshi, Z. Zhou, W. Kellerer, and G. Carle, "A Framework for Reproducible Data Plane Performance Modeling," in *Proceedings of the Symposium on Architectures for Networking and Communications Systems*, ser. ANCS '21, Layfette, IN, USA: Association for Computing Machinery, 2021, pp. 59–65, ISBN: 9781450391689. DOI: 10.1145/3493425.3502756. [Online]. Available: https://doi.org/10.1145/3493425.3502756.

[38] H. Harkous, M. Jarschel, M. He, R. Priest, and W. Kellerer, "Towards Understanding the Performance of P4 Programmable Hardware," in *2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, 2019, pp. 1–6. DOI: 10.1109/ANCS.2019.8901881.

[39] M. Pudelko, P. Emmerich, S. Gallenmüller, and G. Carle, "Performance analysis of vpn gateways," in *2020 IFIP Networking Conference (Networking)*, IEEE, 2020.

[40] M. M. Hasan and H. T. Mouftah, "Latency-aware segmentation and trust system placement in smart grid SCADA networks," *IEEE International Workshop on Computer Aided Modeling and Design of Communication Links and Networks, CAMAD*, pp. 37–42, 2016, ISSN: 23784873. DOI: 10.1109/CAMAD.2016.7790327.

[41] I. 2. Agilent Technologies, "The Journal of Internet Test Methodologies," [Online]. Available: https://web.archive.org/web/20130127153505/http://www.ixiacom.com/pdfs/test_plans/agilent_journal_of_internet_test_methodologies.pdf (visited on 06/02/2022).

[42] *Intel Atom® x7-E3950 Processor*. [Online]. Available: https://www.intel.de/content/www/de/de/products/sku/96488/intel-atom-x7e3950-processor-2m-cache-up-to-2-00-ghz/specifications.html (visited on 06/05/2022).

[43] (2019). Hirschmann EAGLE40, [Online]. Available: https://www.belden.com/hubfs/emea/resources/Picture%20Park%20Assets/Download%20Center%20Assets/EAGLE40-industrial-firewall-hirschmann-11-2019.pdf (visited on 06/05/2022).

[44] P. Emmerich. (May 2022). MoonGen Packet Generator, [Online]. Available: https://github.com/emmericp/MoonGen.

[45] O. E. A. JOHN Samson Mwela, "Impact of Packet Losses on the Quality of Video Streaming," Master Thesis Electrical Engineering, Blekinge Institute of Technology, 2010. [Online]. Available: `https://www.diva-portal.org/smash/get/diva2:831420/FULLTEXT01.pdf` (visited on 05/22/2022).

[46] R. Pauliks, I. Slaidins, K. Tretjaks, and A. Krauze, "Assessment of IP packet loss influence on perceptual quality of streaming video," in *2015 Asia Pacific Conference on Multimedia and Broadcasting*, 2015, pp. 1–6. DOI: `10.1109/APMediaCast.2015.7210275`.

[47] *What is the netfilter.org project?* [Online]. Available: `https://www.netfilter.org/` (visited on 05/25/2022).

[48] *Access Control Lists (ACLs) with FD.io VPP.* [Online]. Available: `https://fd.io/docs/vpp/v2101/usecases/acls.html` (visited on 05/26/2022).

[49] *FD.io - Vector Packet Processing.* [Online]. Available: `https://fd.io/docs/whitepapers/FDioVPPwhitepaperJuly2017.pdf` (visited on 06/02/2022).

[50] J. Falk, D. Hellmanns, B. Carabelli, N. Nayak, F. Dürr, S. Kehrer, and K. Rothermel, "NeSTiNg: Simulating IEEE Time-sensitive Networking (TSN) in OMNeT++," in *2019 International Conference on Networked Systems (NetSys)*, 2019, pp. 1–8. DOI: `10.1109/NetSys.2019.8854500`.

List of Figures

List of Tables

List of Abbreviations

ACL	Access Control List
ASIC	Application-Specific Integrated Circuit
COTS	components-off-the-shelf
CPU	central processing unit
DoS	Denial of Service
DPDK	Data Plane Development Kit
DPI	Deep Packet Inspection
DUT	Device Under Test
GPR	Gaussian processes regression
ICS	Industrial Control System
IDS	Intrusion Detection System
IMIX	Internet Mix
IP	Internet Protocol
IPS	Intrusion Prevention System
IQR	inter-quartile range
NeSTiNg	Network Simulator for Time-Sensitive Networking
NGFW	Next-Generation Firewall
NIC	Network Interface Card
OMNeT++	Objective Modular Network Testbed in C++
PMD	Poll-Mode-Driver
pps	packets per second
SCADA	Supervisory control and data acquisition
SQL	Structured Query Language
SSH	Secure Shell
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TSN	Time-Sensitive Networking
UDP	User Datagram Protocol
URL	Uniform Resource Locator
VLAN	Virtual Local Area Network

VPN Virtual Private Network

VPP Vector Packet Processing