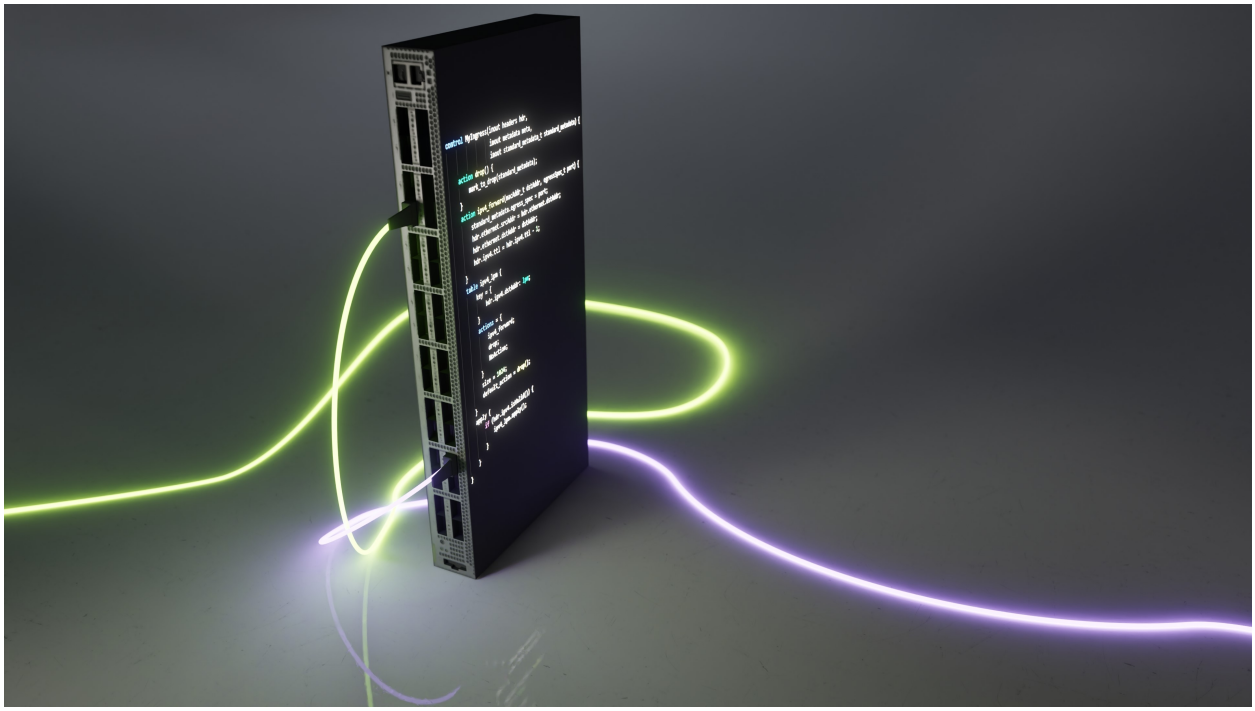
CHALMERS
UNIVERSITY OF TECHNOLOGY



High-Speed, Low-Latency, and Secure Networking with P4

A study of the programming language P4 and it's potential use at Saab Surveillance

Master's thesis in Communication Engineering

Oskar Claeson & William Kruse

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021

MASTER'S THESIS EENX30

High-Speed, Low-Latency, and Secure Networking with P4

A study of the programming language P4 and it's potential use at Saab Surveillance

Oskar Claeson & William Kruse

Department of Electrical Engineering
Division of Communications, Antennas, and Optical Networks
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2021

High-Speed, Low-Latency, and Secure Networking with P4
A study of the programming language P4 and it's potential use at Saab Surveillance
Oskar Claeson & William Kruse

Supervisor: David Olsson, SAAB Surveillance, X Innovations Lab.
Supervisor: Carl Kylin, Division of Communications, Antennas, and Optical Networks.
Examiner: Erik Ström, Division of Communications, Antennas, and Optical Networks.

Master's thesis EENX30
Department of Electrical Engineering
Division of Communications, Antennas, and Optical Networks.
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone +46 (0)31 772 1000

Cover: Graphical representation of a P4 enabled switch, created in Blender by William Kruse

Typeset in LaTeX
Gothenburg, Sweden 2021

High-Speed, Low-Latency, and Secure Networking with P4
A study of the programming language P4 and it's potential use at Saab Surveillance
Oskar Claeson & William Kruse
Department of Electrical Engineering
Chalmers University of Technology

Abstract

The scientific world is rapidly evolving, pulling society more and more toward digitalization. This puts pressure on the digital infrastructures such as communications networks. Not only are the requirements more demanding concerning latency and bandwidth with modern machine- and deep-learning solutions, but the increasing threat in cyber-security also demands safer and more robust network solutions.

In this project, both a literature review and experimental study of the programming language P4 is made. The purpose being investigating if P4 can help optimize network solutions regarding the more demanding requirements and its potential use in Saab's radar solutions. Three concepts utilizing P4 were experimented with, evaluated, and discussed. First, the concept of modification and possibility to write custom-made protocol stacks was evaluated through creating a custom protocol stack, denoted *Shorternet*, which modified a standard link-layer protocol as well as completely removing the internet layer. Second, In-Band-Network-Telemetry, a way to gather data about the network usage and traffic was implemented where the last switch in a flow created a telemetry report with information derived from the switches that were traversed. Third, a data plane firewall was added to switches inside a network which denied access to network devices that were not part of the initial network. Shorternet provided promising results with not only increased effective bandwidth, mainly for smaller sized packets, but also reduced latency and jitter. In-Band-Network-Telemetry proved successful but the increased overhead introduced a significant trade-off with bandwidth and latency. The firewall was easily implemented within the switches and managed to block access at link-layer level through the use of one single rule. This showed potential for further expansion and suggests that the addition of a controller could provide easier management and flexibility. P4 showed great potential on several areas and may well become a staple in modern networking solutions with its added flexibility for development.

Index Terms: Data plane programming, P4, TCP/IP, Network Telemetry, Congestion, Bandwidth, Latency, Security, Software Firewall.

Acknowledgements

We would like to give our thanks to SAAB Surveillance and Chalmers University of Technology for the opportunity to do this master thesis.

We would also like to give thanks to a couple of people at SAAB Surveillance. David Olsson, our supervisor, for the help and guidance during the project and Sven Nilsson for providing the support and resources needed to complete the project. Also, to our supervisor at Chalmers, Carl Kylin, we give thanks for the valuable help with the thesis writing.

Oskar Claeson & William Kruse, Göteborg, August, 2021

List of abbreviations

This section contain a list of the most commonly occurring abbreviations from the report, in order to be able to familiarize with them beforehand and also to be able to fall back upon should it be forgotten.

Abbreviation	Term
Bmv2	Behavioral Model V2
CC	Congestion Control
DNS	Domain Name System
IETF	Internet Engineering Task Force
INT	In-Band Network Telemetry
IP	Internet Protocol
MAC	Media Access Control
MTU	Maximum Transmission Unit
PISA	Protocol-Independent Switch Architecture
PSA	Portable Switch Architecture
SDN	Software Defined Network(ing)
TCP	Transmission Control Protocol
TTL	Time-To-Live
UDP	User Datagram Protocol

Contents

1

Introduction

The project was performed together with Saab Surveillance and evaluated the idea of implementing P4 programs in network infrastructure to optimize traffic flow and improve security within Saab's radar solutions.

1.1 Background

There is a notable increase of interest within the scientific world concerning the use of as much of the raw data provided as possible. This is a considerably different approach compared to the previous more traditional approach, which focus on extracting the relevant data and then discard the rest. Signal and data processing in radar systems are no exception to this, because with the additions of modern machine- and deep-learning solutions the amount of transmitted data is increasing. For Saab Surveillance this has led to new demands on their networking infrastructure, since it must be able to handle massive amounts of data with low latency while retaining high levels of security. In order to update this infrastructure, a possible solution could be to implement custom-made protocols using a new and upcoming domain specific programming language known as P4. The previous standard way of implementing network infrastructure involves a bottom-up design where the hardware defines the network functions due to its limitations. With P4, however, it is possible to use a more beneficial approach of top-down design where the network is instead designed by what is wanted or needed by the developers rather than being limited by the hardware. Multiple network hardware manufacturers have already launched programmable hardware and are continuously developing new versions. This has enabled the use of P4 to create custom-made networking solutions deep in the TCP/IP stack, which may lead to improved performances of future computer networks.

1.2 Scope

The report aims to investigate and evaluate whether or not the programming language P4 can optimize and secure network traffic in radar systems. Custom-made P4 protocols/applications are tested, evaluated, and discussed with regards to bandwidth, latency, jitter and security while compared to traditional TCP/IP networks. Furthermore, a literature review is made to research potential applications of P4. The project is limited to compare custom-made protocols with TCP/IP networks based on Ethernet (802.3-2018) and IPv4. The layers that are investigated during the project are the transport-, network- and link-layers. The project is also limited to software based solutions.

1.3 Ethics

The purpose of the project is to evaluate P4, which is a programming language for software defined data planes in computer networks. Traditionally, the data plane functionality is hardware defined and in order to implement new network functionalities, network devices need to be replaced. With software defined data planes there may be less occasions where there is a need to replace network devices, which could results in less electronic waste. At the same time, in order to obtain a software defined data plane functionality in a traditional network the corresponding network device or devices needs to be replaced.

Regarding the method in this project, there are no clear ethical problems at hand. The project is entirely based on simulations and emulations of network functionality, thus, no one is affected during the method nor is any waste produced.

The results from this project may lead to further studies of software defined data plane programming as the intended purpose of this project is to evaluate if P4 can be utilized to improve network performance and security. Since this project is carried out in collaboration with the defense company Saab AB, there may be derived works intended for military purposes from this project.

1.4 Related research

As P4 is a relatively new and trending language seemingly becoming the de-facto language for data plane programming there are several research articles, short papers etc being published about P4. The working group developing P4 have released several publications together with specifications of the language. These can be found at [1].

2

Theory

In this chapter some theoretical knowledge related to the project is presented. First, the TCP/IP communication model is explained. Secondly, Ethernet and IPv4 headers are presented explicitly. Lastly, P4 and its components and applications are described.

2.1 TCP/IP model

The TCP/IP model as described by the IETF [2] is layered as shown in figure 2.1. Each layer incorporate different protocols depending on the need of the host. The application layer is the top layer in which user specific protocols reside, e.g HTTPS and FTP but also system functions such as DNS. The transport layer primarily uses two protocols called TCP and UDP. TCP [3] is a reliable connection-oriented protocol which ensures the arrival of the packet, where connection-oriented means that the protocol establishes a connection before sending data. UDP [4] is a "datagram" best-effort transport service that strive to minimize the protocol overhead, thus providing a fast but unreliable service. IP is one of the protocols that resides in the Internet layer [2]. IP comes in two versions, IPv4 [5] and IPv6. IPv6 has multiple improvements compared to IPv4, where the most considerable improvements include increased address field sizes and a simpler header format [6]. The link layer [2] houses protocols that provide the required interface to the connected network. Ethernet (IEEE 802.3) is a widely-used link-layer protocol family, which uses MAC-addresses to communicate with other devices.

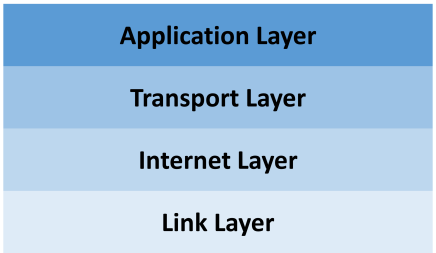


Figure 2.1: *The communication layers of the TCP/IP model. The Link layer is often referred to as layer two (L2), the Internet layer as layer three (L3), and the Transport layer as layer four (L4).*

2.1.1 Ethernet and IPv4 headers

The frame format of the Ethernet standard [7, p. 118] can be seen in figure 2.2. The full packet structure is not described, see the Ethernet standard for details. The frame fields are described as follows in the Ethernet standard:

- The *Destination Address* and *Source Address* fields specifies the destination and source of a packet with a 46 bit long MAC address, which is succeed by two flag bits which totals to 48 bits. In the *Destination* field the first bit indicates if the address is an individual address or a group address. In the Source address field this bit is always 0. The seconds flag bit tells if the MAC address is locally or globally administered.
- The *Length/Type* field specifies the length of the data field or the Ethertype. The value of the field indicates if it should be interpreted as length or Ethertype. If the value is equal or less than 0x05DC the field specifies the length, and if it is equal or greater than 0x0600 it specifies the Ethertype. Ethertype indicates what protocol is above Ethernet in the protocol stack.
- The *Payload* field contains the data.
- The *Pad* field is appended if the frame size is shorter than 64 bytes. Taking the header sizes into account the data need to be larger or equal than 46 bytes.
- The *Frame Check Sequence (FCS)* contains a 4-byte cyclic redundancy check (CRC) value, which is used to verify the received packet is correct.

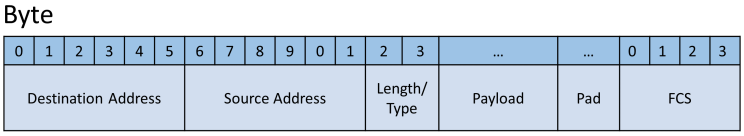


Figure 2.2: *Ethernet frame structure, separated in bytes. The Destination Address and Source Address uses six bytes respectively. Length/Type fields uses two bytes. The payload and pad fields are in variable length. The last four bytes contain the Frame Check Sequence (FCS).*

The IPv4 frame [5], see figure 2.3, consists of several fields and the maximum allowed length is 65,535 bytes but all hosts are only required to be able to accept datagrams up to 576 bytes. The header portion of the IPv4 frame has a maximum size of 60 bytes, but the typical size is about 20 bytes according to [5].

The specific fields and their lengths is defined in [5], which is as follows:

- The *Version* field specifies the version of the IP header.
- The *IHL* field specifies the length of the IP header.
- The *Type of Service* field specifies what quality of service that may be desired by the host. This includes but is not limited to: precedence, delay, throughput and reliability. The use of these

parameters are up to the particular network to implement.

- The *Total Length* field specifies the total length of the header and data in bytes.
- The *Identification* field is assigned by the sender when a fragmentation of the packet has occurred to aid in reassembling the initial packet.
- The *Flags* field includes control flags that indicate, if a packet is allowed be fragmented and if it already is fragmented.
- The *Fragment Offset* field specifies the relative position of the current packet towards the whole fragmentation.
- The *Time to Live* field specifies how many hops the packet has available, this value is set by the sender and then decremented by each hop.
- The *Protocol* field specifies the protocol used for the data in the IPv4 frame, see [8] for the full list.
- The *Header Checksum* field includes a checksum of the header only, which is computed at each hop.
- The *Source Address* and *Destination Address* fields contains the 32-bit long IP-address of the source and destination respectively.
- The *Options* field is a variable length field, which may optionally be used for frames for more functionality. For a full description of the Options field see, [5].
- The *Padding* field ensures that the IPv4 header length is in modulo 32 bits when the *Options* field is present.

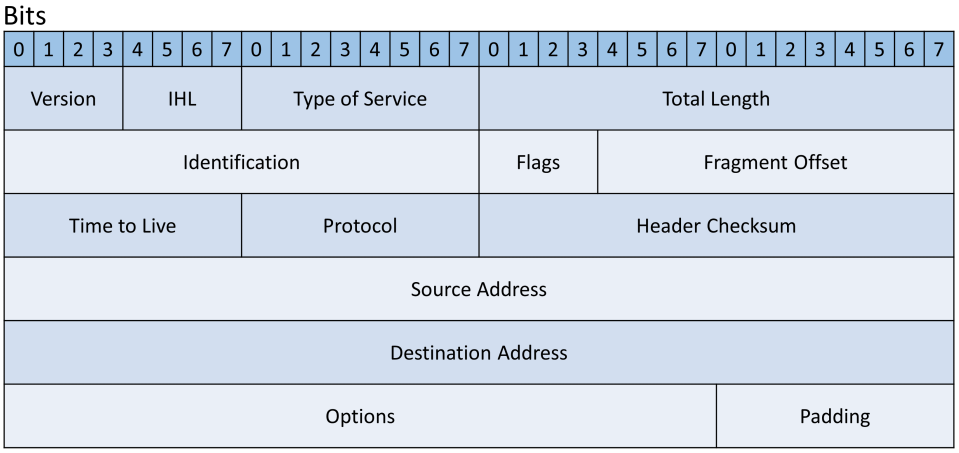


Figure 2.3: *IPv4 header structure, seperated in bits. Each row represents four bytes. The Options field is optional and of varying length. The Padding field is added to ensure that the IPv4 header length is in modulo 32 bits when the Options field is present.*

2.2 P4

The programming language P4, which stands for Programming Protocol-independent Packet Processors, is designed as a means to further improve the functionality of Software-Defined Networking (SDN) by working in conjunction with SDN control protocols. The design of P4 was proposed with three main goals in mind [9]. Firstly, to be able to modify the behavior of network hardware after deployment. Secondly, to introduce a protocol independence of the hardware in order to not to be limited to specific network protocols. Lastly, a target independence where the P4 programmer need not consider the specifics of the hardware that is to be controlled. Currently, there has been two releases of P4 specifications, the first one now known as $P4_{14}$ [9] and an updated version known as $P4_{16}$ [10]. The $P4_{16}$ version intend to stabilize the language using $P4_{14}$ as a fixed core together with a set of specific libraries suitable for evolution as well as other flexibility mechanisms. This library should be included in every p4 program by adding the definition file core.p4.

2.2.1 Architectures

When writing a program in P4, there is a need for a definition of the target in order for a compiler to properly compile the program to the hardware. This is where the concept of architectures comes into play. An architecture, as described in the $P4_{16}$-specifications [10], is a .p4 file definition similar to the core.p4 and is intended for vendors to provide the programmers with in order for them to be able to compile to the hardware. Having access to several architecture definitions can be seen as enabling portability for a P4 program to multiple different targets.

The programmable pipeline of a switch is often referred to on an architectural level as a Protocol-Independent Switch Architecture (PISA). The PISA is based on three parts, all of which are programmable: A parser, a match-action pipeline, and a deparser. The model is explained very simply in figure 2.4. Stephen Ibanez, a PhD student with connections to the P4 consortium, discuss the PISA shortly in the beginning of an introduction video made by the consortium [11]. The parser can be seen as a finite state machine that reads the headers of incoming packets and extracts identifying fields which are to be matched in the next stage. At the match-action pipeline the extracted data is passed through three stages: firstly, a checksum verification which is intended to catch erroneous packets. Secondly, the packets get passed along one or more match-action units, which are programmed to perform a match on the fields and potentially follow up with an action. Thirdly, a checksum update is performed since during the match-action stages there may have been some modifications of the packet which must be updated. Finally, packets arrive at the deparser which re-serialize the headers together with the payload before it is passed through to the output.

To illustrate the workflow through PISA a simple example is presented: a simple IP forwarding procedure. A packet consisting of an IPv4 header over Ethernet frame together with a payload arrives at the parser. The parser extracts the first-most header, which is the Ethernet header and reads the information. One of the fields in the Ethernet header is the type field containing information about the following protocol, in this case IPv4, which tells the parser that the next state is to parse the IPv4 header. After the IPv4 header is extracted then usually any following protocol such as TCP or UDP are parsed, however, in this simplified case this is ignored since only the IPv4 and Ethernet headers are of interest. Instead the packet is directly accepted after IPv4 is parsed and is then passed through to the match-action pipeline. Here the packet together with its metadata is processed and potentially some actions are performed depending on matching tables entries. For instance in this case the forwarding

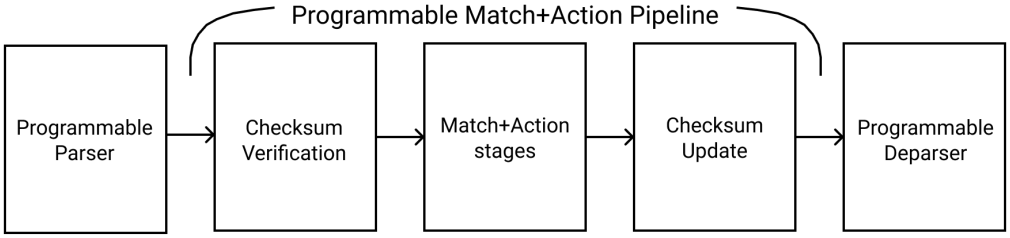


Figure 2.4: *Simple description of the block structure flow for the PISA. First, a packet arrives at the programmable parser and afterwards it is forwarded to the programmable match+action pipeline, which includes a checksum verification, match+action table, and checksum update. Lastly, the packet is forwarded to the programmable deparser.*

is supposed to port the packet to the correct output port connected to the next hop according to the forwarding table lookup. Thus, the matching is based on the IPv4 destination address. Now the packet is either dropped due to a faulty address or the packet is forwarded accordingly. At the deparser the two headers, IPv4 and Ethernet, are then re-serialized in the correct order and passed to the output of the switch ending the forwarding workflow example.

The P4 consortium have been working on a way of abstracting the hardware pipeline, which is needed in order to not be bound to any specific switching chip. The work-in-progress specification for the target architecture is denoted Portable Switch Architecture (PSA) [12], which consists of six P4 programmable blocks together with two fixed-function blocks. The two fixed-function blocks are called "Packet buffer and Replication Engine (PRE)" and "Buffer Queuering Engine (BQE)" both of which are target dependent. Comparing the PSA with the simple PISA, see figures 2.4 & 2.5, it can be noted that the programmable blocks in the PSA can be seen as two PISA structures, one for ingress and the other for egress. What this means is the first section takes incoming packets to the switch and process them before sending them in to the PRE, after the PRE the packets are once again processed before sending them out to the BQE.

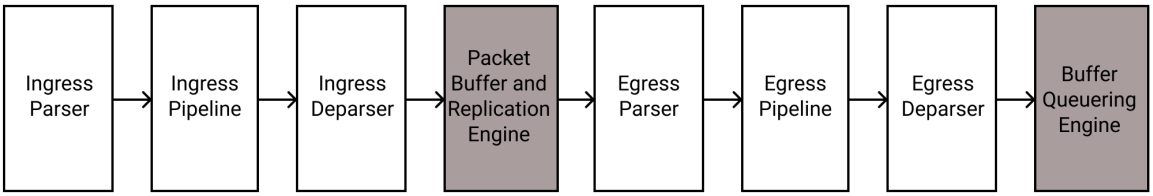


Figure 2.5: *Simple description of the block structure flow for the PSA. A packet first arrives at an ingress process consisting of a parser, match+action pipeline, and deparser. Then it is forwarded to the fixed function PRE. The PRE then forward the packet to an egress process similar to the ingress process before it is forwarded to the fixed function BQE.*

As the P4 consortium is working on the PSA that is to be an one-fit-all solution, a more simple and commonly used architecture by P4 programmers at the moment is the V1model. The V1model architecture is based on $P4_{16}$, but is designed to mimic the functionality of a $P4_{14}$ switch, in order to be able to translate older programs onto the later version. V1model is structured with six programmable blocks and a fixed function traffic manager, which are visualized in figure 2.6. The traffic manager handles packet queuing, replication and scheduling, similar to the fixed function blocks purpose in

the PSA model. The idea behind the V1model is for programmers to be able to utilize and translate $P4_{14}$ programs until the PSA has been fully defined which will provide more functionality. As the V1model is not being focused on by the P4 developers there are some flaws and "illegal" actions that are accepted by compilers that will not properly work in the hardware, which may cause some issues.

Figure 2.6: *Simple description of the block structure flow for the V1model architecture. First, a packet arrives at an ingress process consisting of a parser and match+action pipeline, which is not including the checksum update. Then the packet is forwarded to a traffic manager before it is forwarded to an egress process. The egress process consists of a match+action pipeline without a checksum verification and ends with the packet arriving at the deparser.*

Currently, switch vendors are working on defining their own architectures, for example the company Barefoot (owned by Intel since 2019) have developed their own architecture called Tofino Native Architecture (TNA) for their Tofino switches, which is not described here. In the language specification of $P4_{16}$ [10] it is specified that manufactures of P4 enabled switches are expected to provide an accompanying architecture definition with their product. The reasoning behind this is that vendors can then express the uniqueness of their chips and build upon them for newer releases etc. and not be bound by standards. The architecture definition, however, does not need to express every functionality of the target, the manufacturer may even opt to have several definitions for the same target but with different capabilities.

2.2.2 Behavioural model

The behavioural model version 2 (Bmv2) switch [13] is a reference P4 switch that has been configured to use different P4 targets such as simple_switch (V1Model), simple_switch_grpc (V1Model) and psa_switch (PSA). Bmv2 targets use a .JSON file, which is generated at compilation of a .p4 program to apply the behavior of said program. Mininet, which is a network emulator software [14], can then be used to emulate networks consisting of Bmv2 switches loaded with the programs.

The performance of Bmv2 can potentially be affected be a number of factors [15]. These include, but are not limited to, system hardware, number of match-action tables and entries in the .p4 program installed on the switch, host OS, and the version of the Bmv2 switch. The P4 consortium claim that Bmv2 is rated up to around 1Gb/s. Antonin Bas, a developer for P4, mentions in a github issue [16] that the ingress pipeline is often the bottleneck of the program. They also mention that the throughput linearly depends on the amount of match-action tables.

2.2.3 In-band network telemetry

In-Band Network Telemetry (INT) specification [17] was initially introduced by the P4.org Applications Working Group in 2015. The current version 2.1 was released in 2020. It proposes a new way to gather telemetry data in networks without applying the control plane. This data can be used for multiple applications, which are mentioned in section 2.2.3c. The specification defines data

that can be gathered in each switch, such as node ID, Egress/Ingress interface ID, hop latency and Egress/Ingress timestamps.

There are multiple headers defined in the INT format. INT-shim is a header that indicates which mode is used and what protocol is next in the protocol stack. The different modes are presented in the next section. INT-header can contain information such as version, per hop length of headers and data, remaining hops and instructions. INT-data contain data from each switch respectively. INT report header is a header which precede the telemetry report data. The telemetry report format is specified in [18], which is published by the P4.org Applications Working Group.

There are three different switches defined in an INT-flow. INT-source is the first P4 switch in a flow. The INT-source can add the initial INT-headers and INT-data. Depending on the mode, which is explained in the next sub-section, INT-source sends reports to the monitoring system or adds the initial INT-headers and INT-data to the packets. INT transit-nodes are consecutive P4 switches in the flow, which can read INT-headers and act accordingly. INT transit-nodes send reports directly to the monitoring system or append their data depending on the mode. INT-sink is the last P4 switch in a flow. The INT-sink is able to remove all the potential INT-headers and INT-data and forwards the original packet to the destination.

2.2.3a INT modes of operation

The INT specification [17] defines three modes of operation: INT-XD (eXport Data), INT-MX (eMbed instruct(X)ions) and INT-MD (eMbed Data). A visualization of INT-XD, INT-MX and INT-MD can be seen in figures 2.7, 2.8 and 2.9, respectively. With INT-XD each node in the network sends reports to the monitoring system and the packet from the source host is not altered any point. For INT-MX, an INT header are appended at the source node which include some protocol information and instructions that successive nodes can read and act upon. With INT-MD, an INT header and the data from each node is appended to the packet, a report is then created at the sink and sent to the monitoring system. INT-MD is focused on in this project.

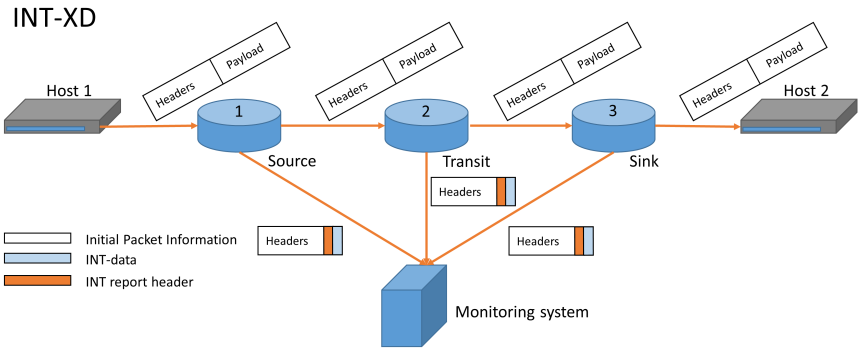


Figure 2.7: *A description of the consecutive packet header formats and how reports are sent to the monitoring system when using the INT-XD mode of operation.*

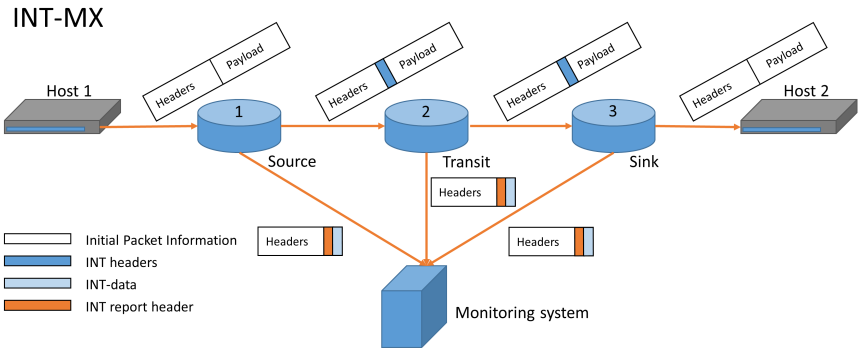


Figure 2.8: *A description of the consecutive packet header formats and how reports are sent to the monitoring system when using the INT-MX mode of operation, where INT headers are added at INT-source. INT headers may include INT-shim and INT-header*

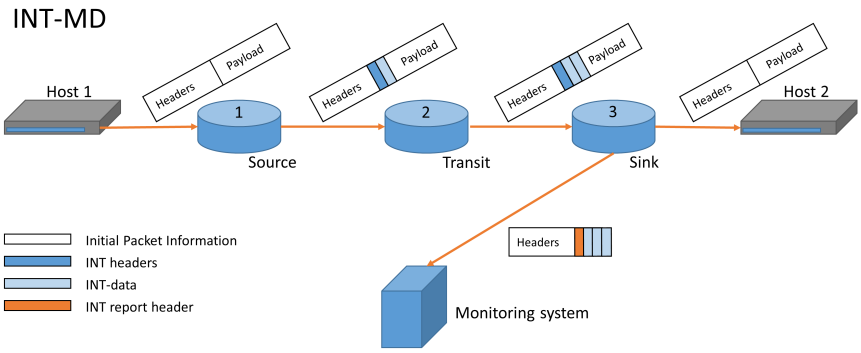


Figure 2.9: *A description of the consecutive packet header formats and how reports are sent to the monitoring system when using the INT-MD mode of operation, where INT headers and INT-data is added to the packet at each hop. INT headers may include INT-shim and INT-header*

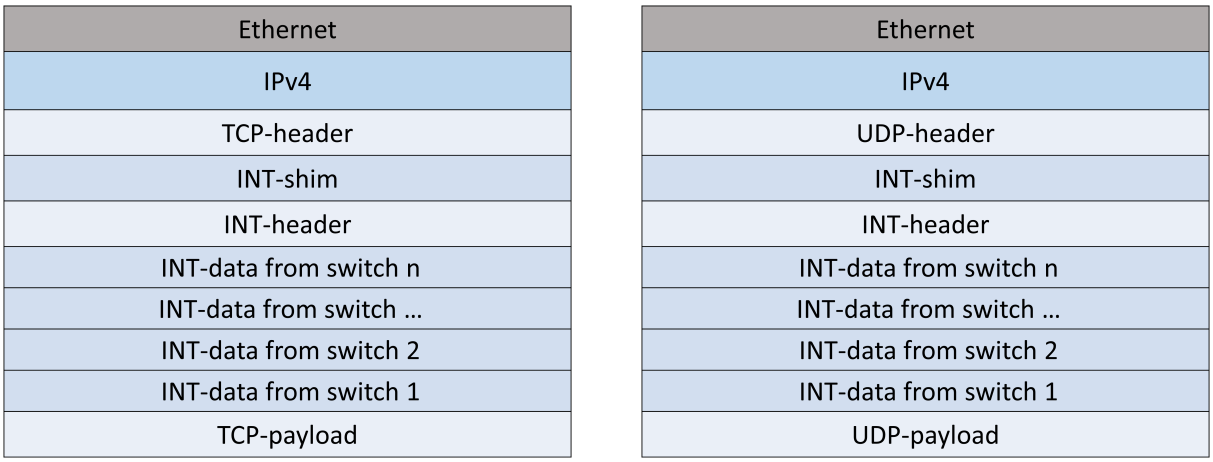
2.2.3b INT-header placement for INT-MD

INT headers and INT-data can be placed at different places in packets. A list of examples is available in the INT-specification [17], however, it is entirely up to the network programmer to decide where INT is placed and what headers to use. In all cases the INT-source adds the INT headers and the headers that correspond to a certain mode of operation to the packet. A couple of commonly used INT-MD placements over TCP/UDP from the specification can be seen in figure 2.10.

In figure 2.10a INT is placed between the TCP-header and TCP-payload. The DSCP flag is set to 0x17 in the IPv4 header to indicate that INT is present. In figure 2.10b INT is placed between the UDP-header and UDP-payload. In this case the destination port field in the UDP-header is used to indicate that INT is present. However, what that value should be is not currently specified in the specification.

The specification uses a couple of different terms than what are used this thesis. INT-metadata in the specification refers to the INT-header combined with the INT-data. INT-header in the specification

also refers to all the INT headers and data i.e INT-shim, INT-header and INT-data combined.



(a) *INT placed between TCP header and payload.* **(b)** *INT placed between UDP header and payload.*

Figure 2.10: *Figures with examples of different INT-MD header placements, as specified in the INT specification [17].*

2.2.3c Applications with INT

It is possible to leverage INT in many different applications. Some examples that are presented in this thesis are Congestion control (CC), load balancing and anomaly detection.

CC aims to limit the bandwidth of the senders to avoid large queues in network devices. Congestion typically occurs when one or several flows exceed the bandwidth limit a device is capable of processing/outputting. When a switch experiences congestion, the latency increases due to longer queues. With CC the queue build-up can be reduced thus reducing latency at the cost of bandwidth. CC was first applied after the internet congestion collapse in 1986 when V. Jacobson released a congestion avoidance algorithm [19]. With today's fine-grained metrics, the reason for congestion and other faults can be pinpointed on a fine level. For example Intel Deep Insight claims it is able to resolve the latency induced on a packet in nanoseconds [20]. Yuliang et al. proposed HPCC: High Precision Congestion Control [21] which leveraged INT with P4. They implemented a new CC algorithm and showed results of 95% less flow-completion time compared to DCQCN [22] and TIMELY [23]. DCQCN and TIMELY are previous congestion control systems in Remote Direct Memory Access (RDMA) networks. Flow-completion time refers to the time it takes for a flow of multiple packets to traverse the network. The authors implement a custom form of INT where the sink sends back a report to the initial sender. The sender can then use this information to control its bandwidth. By default the algorithm causes a 5% loss in maximum bandwidth, which results in almost zero queues. The maximum bandwidth is defined as the bandwidth of the link with the lowest bandwidth capability, e.g bottleneck link. The bandwidth loss could be configured if bandwidth is crucial, but the sent bandwidth can not exceed the bandwidth capability of the bottleneck link. The HPCC protocol appends a header of 2 bytes at the source and 8 bytes of INT data is added at each hop. The algorithm was designed to improve the congestion algorithms already present in RDMA over Converged Ethernet Version 2 networks.

Another form of congestion control is load balancing, however, the difference is that with load

balancing the flows are separated between switches. A comparison can be that two traffic lanes are used instead of one. J. Kim et al. implements load balancing with INT [24]. Each switch gathers and saves network metrics with INT. Then the switches route flows accordingly. They compare it to Equal-Cost Multi-Path (ECMP) [25] routing and conclude that their system has lower throughput and longer flow-completion time, when congestion is not present in the network. They explain that this is due to the increased overhead. However, during congestion the throughput is significantly higher and the flow-completion time is lower.

Load balancing can also be applied by a controller, J. Hyun, N. V. Tu and J. W. Hong discusses and implements parts of knowledge-defined networking [26]. The idea is to feed the data generated by INT to machine learning algorithms in a knowledge plane. Insights from the knowledge plane can then be handed over to the control plane which can take appropriate actions in the network. The authors argue that, the control plane have difficulties handling the large amount of INT data alone. The knowledge plane should thus run on another machine to offload the controller. The authors claim that this system can be used for traffic engineering and anomaly detection.

2.2.4 Firewall

The purpose of a firewall is to filter out unwanted or harmful traffic to provide security within the network. Generally, a firewall sees usage in internet- and transportation-layers of the TCP/IP model, but can also be able to analyze application level traffic. Traditionally, firewalls are either hardware or software based. A hardware based firewall, also known as perimeter firewalls, protects the network and the traffic going in or out from the perimeter. A network may have several hardware firewalls within the network and not solely at the edges to create more boundaries and separate accesses within the network. This is useful for managing and controlling how the network functions, but it also provides security for devices without built-in firewalls, for example printers. Software firewalls on the other hand works like a backup to individual systems for when a breach in the network has already occurred. The software firewall may detect these breaches by consulting a database, which indicates whether the traffic is of a malicious nature or not. With the rise of SDN and virtualization, physical hardware firewalls can instead be virtualized as software implementations to safeguard cloud-based networks or virtual networks to provide more security. An example of a virtual hardware firewall is *VNGuard*, proposed by Deng et al. [27], which adaptively finds and places virtual firewalls at entry points. Hu et al. proposed another SDN motivated firewall called *FlowGuard* [28] which intends to provide firewall protection for OpenFlow-based networks. This, however, is required to follow the constraints of OpenFlow protocols. With the usage of the P4 language, a firewall can be programmed directly on P4-enabled switches, allowing for custom network solutions to be protected at the data plane level. This may also provide firewalls within the network, which can protect against potential attacks launched within the network. One such firewall implementation is called *P4Guard*, by Datta et al. [29]. Datta et al. state that P4Guard is an easily configurable software implementation that can update functionalities and install dynamic firewall rules on-the-fly through data plane programming utilizing the P4 language. P4Guard leverages a controller which are able to both add and remove software firewalls dynamically through the network. P4Guard is based on the Bmv2 switch which limits the performance compared to VNGuard, in a similar fashion as that described in section 2.2.2.

3

Method

This chapter presents the approach and methods used to fulfill the scope of the project. The intention is to also provide a base upon which readers may understand and replicate the tests and provided results. First, the testing environment used in the project is described. Second, an experiment to test the use case of removing redundant information and how it was performed is explained. Third, the concept of INT and how it was implemented during the project is described. Lastly, a proof of concept firewall idea is presented and described.

3.1 Testing environment

To be able to conduct the tests and evaluate the programming language and its potential, a virtual environment was constructed. The environment was emulated using Mininet and consisted of Bmv2 simple_switch_grpc switches that were loaded with P4 software. No SDN controller was used, instead table entries were added to the switches at startup with simple_switch_CLI. New entries could also be added at runtime with simple_switch_CLI if needed. Mininet runs on a virtual machine running Ubuntu 18.04 with 8 Gb of ram and 4 CPUs. The host machine runs Ubuntu 20.04 with an Intel i7-10510U processor and 16Gb of 2667 MHz RAM. Versions of the different software that were used were, Bmv2 (1.14.0), Mininet (20.3.0b2), protobuf (3.6.1), p4c (at commit 24895c1), and PI (at commit c65fe2e).

3.2 Removing unused information in packets

One of the potential use cases of P4 is the flexibility to easily change, add new, and remove protocols from the standard protocol stack of a packet-switching network. In a high performance application where every byte counts, this flexibility could potentially improve the efficiency of the traffic by removing redundant information. One such case that was tested was to modify the standard layer two Ethernet frame as well as completely removing the layer three protocols. Ethernet was chosen as this is the most commonly used standard for link-layer communication, especially concerning IP packets. The motivation behind the modifications were that in an air gapped and statically defined network with limited switches and hosts there may not be a need to use the full MAC addresses to be able to uniquely identify the network devices. Furthermore, if the network structure is known, then a routing process, which utilizes IP, is no longer needed. The modified protocol stack, which is denoted by the authors as *Shorternet*, together with the standard TCP/IP stack are visualized in figure 3.1.

With the idea to bypass standard IPv4/IPv6 protocols, the layer four protocol was added directly on the modified layer two protocol instead. This way the header size of the packets is reduced and leaves more space for raw payload. However, with this removal the packet no longer have a Time-to-Live (TTL) field nor a transport layer type field, which usually reside in the IP protocol. Therefore, the TTL field is included in the new layer two protocol in order to secure the network from endlessly propagating packets, as well as changing the standard etherType field to address the layer four protocols instead. The previous size of the Ethernet frames were two MAC address fields of 6 bytes each and the etherType field of two bytes. The new and modified layer two protocol instead has four single byte fields, two address fields, a new protocol field, and lastly, the TTL field. Thus the new layer two protocol has been reduced from 14 header bytes to 4 header bytes. With the removal of the IP protocols, at least another 20 bytes have been removed from the protocol stack (due to optional headers this can be larger). In total a minimum of 30 header bytes are removed from each packet, see figure 3.1. The reason why for example the TTL field is 1 byte when it is enough to have 4 bits or less is because when using the V1model, some limitations of the older version $P4_{14}$ is retained, where the total header size need to be in full bytes.

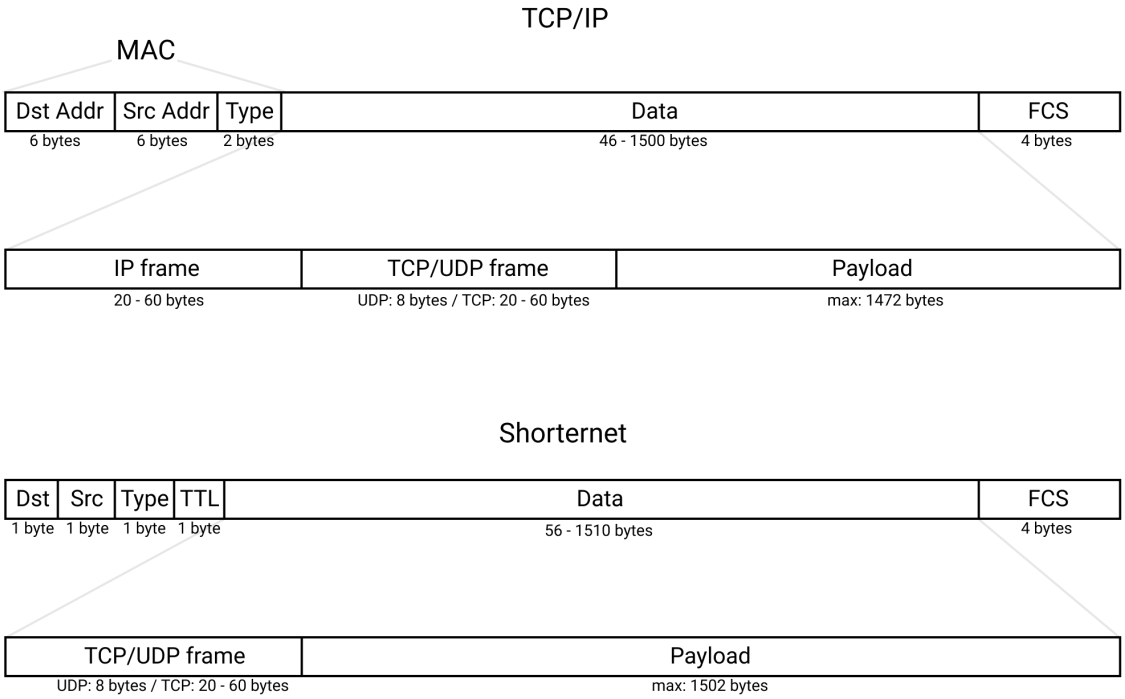


Figure 3.1: *Simple representation of the differences between Standard TCP/IP protocol stack (top) vs the Shorternet protocol stack (bottom). The MAC part of the TCP/IP structure has been reduced and includes a TTL field instead in the Shorternet structure, leaving more room for the Data section. The IP frame has been removed in Shorternet, leaving more room for Payload*

The potential gain in effective bandwidth should reduce as the packet length increases. In order to prove this hypothesis the payload length was increased during benchmark tests to gather insight of how the payload length affect the results. The tests were performed on an emulated network using Mininet as described in section 2.2.2. The structure of the network is visualized in figure 3.2 where Host 1 is the sender and Host 4 the receiver. The test consisted of a single flow traversing the network of 100 000 packets per flow with varying packet lengths.

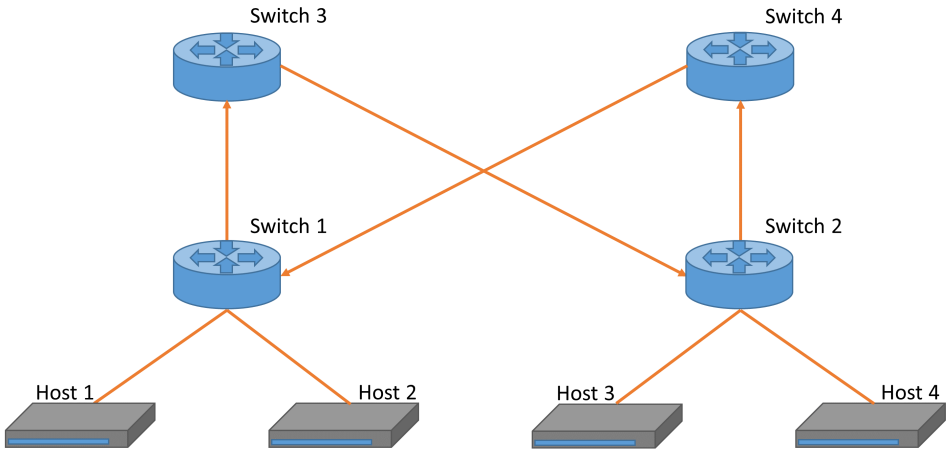


Figure 3.2: *Structure of the network used for the benchmark test of Shorternet vs Standard TCP/IP protocol stack. The path that packets traversed during the tests was: Host 1 → Switch 1 → Switch 3 → Switch 2 → Host 4.*

3.2.1 Benchmark test

In order to create a fair comparison between the benchmark-tests of the standard and Shorternet protocol stack, custom test scripts were written. These were needed as the existing tools to measure network statistics such as received bandwidth, latency, jitter, and packet loss are not compatible with the custom written Shorternet protocol stack. The benchmark test scripts are written in Python2.7 and utilize sending timestamps and sequence numbers together with a string of randomized text as raw payload. At the receiving side these are then used to calculate the network performance metrics. The flowchart of these scripts are presented in figure 3.3

The idea of the test scripts were to create packets of equal packet length where 100 000 packets are sent with a similar packet rate to ensure fairness. Due to the fact that the network is emulated, the performance is potentially reduced by a number of factors as mentioned in section 2.2.2. In early tests it was concluded that on the test setup the bandwidth of the network was at the highest at around 40Mbit/s. This bandwidth reduced even further when adding more switches to the emulation. To ensure the network could keep up with the flow while still retaining fairness, the packet rate was fixed to approximately 1000 packets per second (pps) during all tests using a sleep function in the scripts. By tweaking the sleep function, the packet rate could be changed to a slightly higher value and still see good performance from the switches. However, due to the way packets were created with Scapy, which is a program in python that enables packet manipulation [30], the fairness was compromised when tweaking the sleep function because then the packet rates were no longer similar between the two protocol stacks. Therefore, the packet rate of 1000 was used. The tests were conducted by sending 100 000 packets for each packet length. This gives a large enough sample base of which accurate averages and standard deviations could be derived.

Packet loss is calculated via the sequence numbers, where after a packet has been received the sequence number is read and then checked if this matches the expected number. If the received number is higher than the expected one then packet loss has occurred, which is kept track of with a

counter. At the same time, if a received sequence number is lower than the expected then the packet is late and the loss counter is updated since this packet is no longer lost but is instead out of order. Neither the sender nor the switches are able to resend the same packet, thus making sure no duplicate packets traverse the network. The latency metric is modeled as a stochastic variable named L, which is defined as the time-difference between the received timestamp of the sent packet and a timestamp taken at the receiver side when a packet has arrived:

$$L = t_{\text{arrival}} - t_{\text{transmitted}}. \quad [\text{ms}] \tag{3.1}$$

The arrival timestamp is taken directly on arrival in order to not include the processing time of the script itself. A latency measurement, $l_i, i \in [1,n]$, is generated for each packet that arrives at the receiver. The latency for a specific packet length is estimated as an average over all n measurements according to:

$$\mu_L = \frac{\sum l_i}{n}, \quad [\text{ms}] \tag{3.2}$$

where n in this case is 100 000 packets minus the number of packets lost. If a packet has been lost, the latency of that packet is simply not included, thus, the number of packets lost is to be presented together with the latency. There are different ways to define jitter, for computer networks it can be seen as a metric describing variations in latency and is sometimes referred to as Packet Delay Variation. In this test, during each iteration it is assumed that the network channel is not varying with time. For instance, the channel does not need a stabilization period and the transmission flow is neither bursty nor varying in size due to how the test is carried out. Therefore, jitter is defined as the standard deviation σ of the latency and is estimated according to:

$$\sigma_L = \sqrt{\frac{\sum (l_i - \mu_L)^2}{n-1}}. \quad [\text{ms}] \tag{3.3}$$

The final metric that is estimated is the received effective bandwidth. This was done via the following equation:

$$B_{eff} = \frac{\text{bytes}_{\text{tot}}}{t_{\text{flow}}} \cdot \frac{\text{payload length}}{\text{packet length}}, \quad [\text{bytes/s}] \tag{3.4}$$

where a counter, $bytes_{tot}$ keeps track of the number of processed bytes as well as using a first arrival timestamp and last arrival timestamp to find the total flow time, t_{flow}. The payload length and packet length variables indicate the length of the raw payload of packet in bytes and the total length of a packet in bytes respetively.

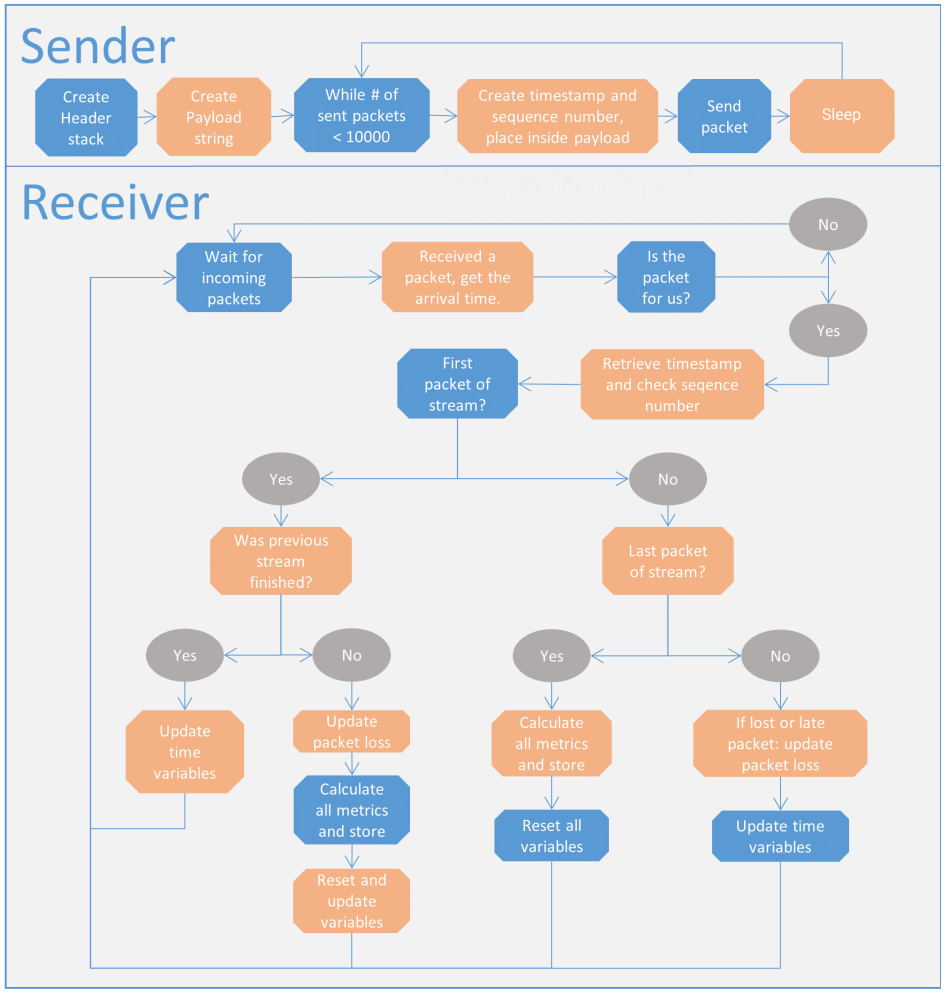


Figure 3.3: *Flowchart describing the logic of the scripts used for the sender and receiver during the benchmark test implementation. In this case Host 1 ran the Sender script to send 100 000 packets with an artificial packet rate created via a sleep call in the end of the loop. Host 4 ran the Receiver script, which listens for incoming traffic and if a correct packet is received it is then processed accordingly*

3.3 In-Band Network Telemetry

A network with INT was implemented and compared against a traditional network without INT in order to gather insight into how INT affects the network. Bandwidth, latency, and jitter was compared between the two networks.

The implemented INT protocol follows the structure of INT-MD over UDP as specified by the INT specification [17]. Focus on applying INT to UDP traffic was chosen because applications in radar systems often use UDP or similar best effort protocols to stream data from the antenna-arrays. The INT protocol was constructed with CC in mind, the goal being to achieve high bandwidth networks while maintaining low-latency. However, there was not enough time to properly implement and evaluate a CC algorithm. In contrast to how INT reports were forwarded in the INT specification where reports were sent to a monitoring system, the INT report generated by the INT-Sink is sent back to the sender instead. This is done because the sender should apply CC on its own output, which uses INT-Data, and in a sense becomes the monitoring system for its own flows. The INT-shim header was not used at all. This was possible because only INT-MD over UDP was present in the network. If other INT protocols were to to be used, INT-Shim would be needed to separate these.

The INT-header format can be seen in figure 3.4. This header is added at the INT-source. The *hop count* specifies how many hops an INT packet has traversed, this field is incremented at each hop. *Hop count* is used when parsing INT-Data at consecutive nodes. The UDP destination port number is copied to the *UDP Port* field from the UDP header. The INT-source changes the UDP destination port number to a known number to indicate that INT is present. The INT-sink uses the saved UDP port number when reconstructing the original packet. The INT report header uses the same format as the INT-header.

The INT-data header structure can be seen in figure 3.5. This header is appended to the packet at each hop. *Timestamp* saves the egress timestamp. *Transmitted bytes* saves the total sent bytes from the egress port. *Queue length* saves the queue length at the egress port. *Switch ID* saves the id of a switch. *Port* saves the egress port the packet is forwarded through.

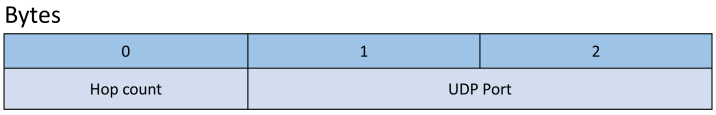


Figure 3.4: *Structure of the implemented INT-header fields. The first byte contains the Hop count and the remaining two bytes contain the UDP Port. The total size of the INT-header header is three bytes long.*

The system was then tested in two tests, where both were conducted using a linear network topology. In the first test the diameter of the network was increased to gather insight how the flow length affects the network. In the second test one additional UDP-flow is introduced to the network to gather data on how multiple flows affect the network.

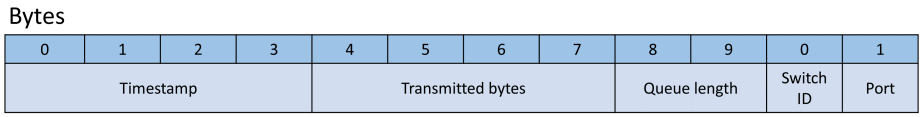


Figure 3.5: *Structure of the implemented INT-data fields. The first four bytes contain the Timestamp, bytes 4-7 contain the value of total Transmitted bytes. Bytes 8-9 represent the Queue length and the last two bytes represent the Switch ID and Port respectively. The total size of the INT-data header is 12 bytes long.*

3.3.1 Switch logic

All of the following logic assume that the packet being processed is a UDP-packet. Otherwise the process follows a basic forwarding logic. In figure 3.6 the complete flow of a packet through a switch is visualised. Each switch parses all the present headers in the packet, if INT-data is present the payload is then parsed as a variable header. This enables the switch to remove the packet data later on when creating a report if the switch is a sink. At the end of the ingress pipeline the switch checks if it is the last one in a flow. This is done via matching the destination address against a table and if it is a match, the packet is cloned. The clone is used to create an INT report in the egress pipeline. The packets are then passed to the *traffic manager* and then to the egress pipeline.

The egress pipeline has two different parts, one for a potential cloned packet and one where INT-data and/or INT-header is to be appended. If the packet is not a clone the switch continues to check if the switch is a sink. If it is not a sink then the INT-header and INT-data is added as usual. Should the switch be a sink, the INT-header and data is removed to ensure that INT is not present at the receiving end for the original packet. If the packet is a clone, INT-data is added from the sink switch. Then the payload is removed and the report is forwarded towards the host that sent the packet in the first place.

In section 2.2.3b it is mentioned that the UDP destination port is used to indicate INT in a packet. However, the value of this parameter is not decided by the specification. The port number to indicate that INT-metadata is present in a packet was chosen as 1337 and the port number to indicate that the packet is a report was chosen as 1338. This information is used in multiple places in the logic. In the parser the UDP destination port is checked to indicate if it should parse INT-header and data. Furthermore, if the packet is an report then INT-data should not be appended to that packet.

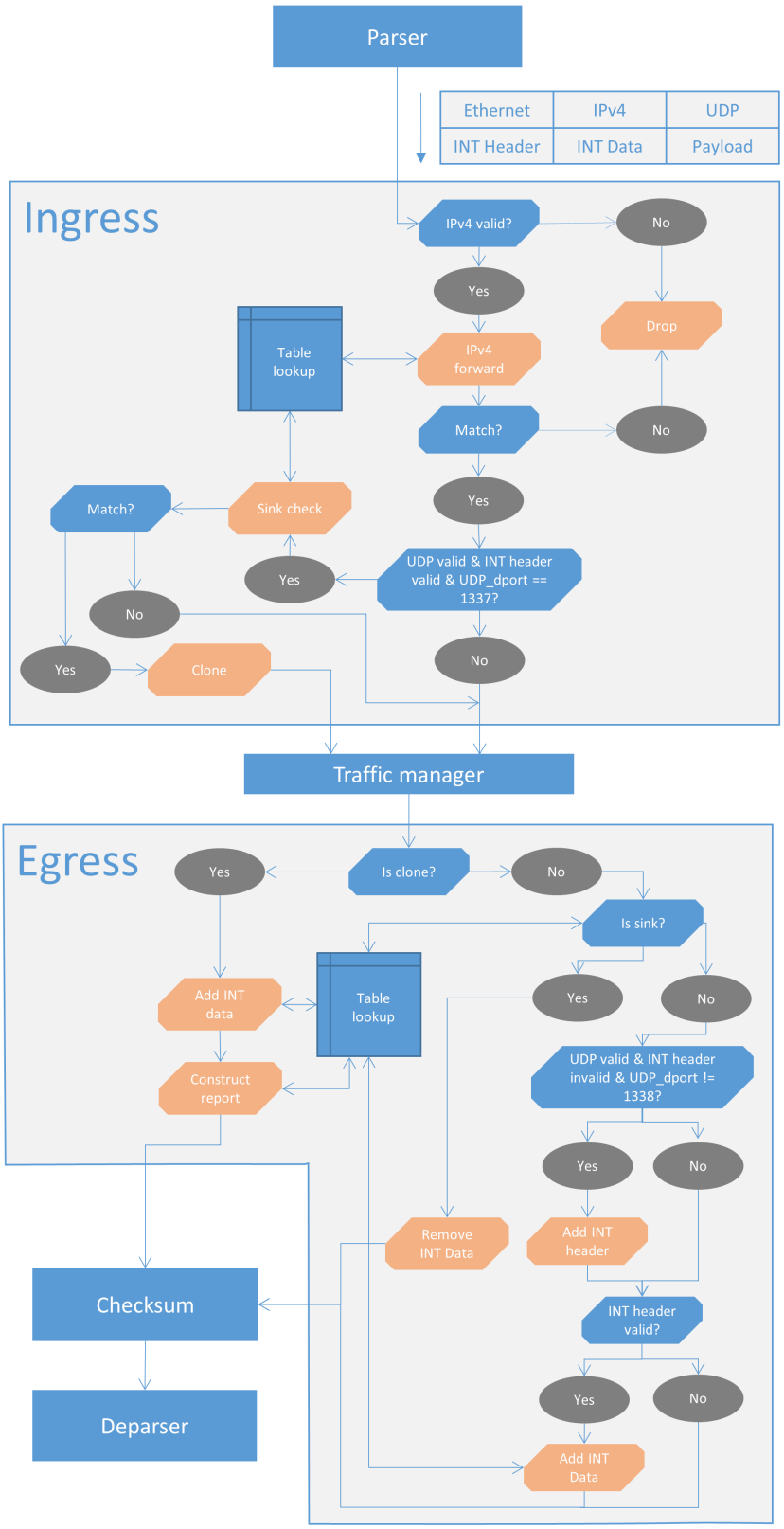


Figure 3.6: *Flowchart of the P4 switch logic for the INT implementation. The ingress block describes the ingress pipeline for INT. If cloning occurs, two packets are sent out of the ingress pipeline. The egress block describes the egress pipeline for INT. The results are then sent to a checksum updater, and lastly, to the deparser. If cloning occurred in the ingress pipeline the egress pipeline is invoked twice for one ingress packet.*

3.3.2 Benchmark

In the first test with increasing network diameter, three sub-tests were performed using three, four and five switches respectively. The topology with four switches can be seen in figure 3.7. A UDP flow was introduced with *iperf2* [31]. *iperf2* is a free software network measurement tool for measuring bandwidth, packet loss, and jitter. Specifically, the jitter measurements of *iperf2* are, in contrast to Shorthernet, done as specified in section 6.4.1 in the RFC3550 [32]. The jitter, in this case, is estimated at every new received packet together with a weighted value from the previous estimation. The result is a smoothed value of the deviation of the latency between packets. This method is more accurate for a channel varying with time, which is the case with INT. This is partly due to that there is a stabilization period during the start of a flow since the report flow start after the first packet arrives at the last switch, as well as that congestion is expected to occur.

The payload length was fixed to ensure that the Maximum transmission unit (MTU) was not exceeded in any of the topologies. For each topology, multiple tests with an increase in bandwidth was conducted to detect when congestion occurred in the network. The performance of the network is reduced with an increasing number of switches, as mentioned in section 2.2.2. Because of this, the maximum bandwidth was configured to 6 Mbit/s for all hosts and switches in order to reduce the effect of limited system performance. The tests ran for 120 seconds for each bandwidth, split into six 20 seconds sub-tests. This means each flow was 20 seconds long, which ensures that congestion occurs if the switches are not able to handle the incoming bandwidths. If flows are too short, packets only experience longer queues and no packets are dropped.

In the second test the topology was fixed at four switches. The bandwidth was also limited to 6 Mbit/s to ensure fairness. In this test the bandwidth was increased as in test one, but at the same time another *iperf2* UDP flow was introduced between two other hosts at a fixed bandwidth of 2 Mbit/s. This was done to gather better insight into how multiple flows with INT affect the network, since there is seldom one single flow residing in the network at a time. The flows are visualized with black arrows in figure 3.7. The test ran for 120 seconds for each bandwidth, split into six, 20 seconds sub-tests.

In both tests the metrics were gathered from an *iperf2* server which was active at the receiver. Latency was gathered in both tests by using *ping* [33]. The interval for *ping* was configured to 0.1 seconds.

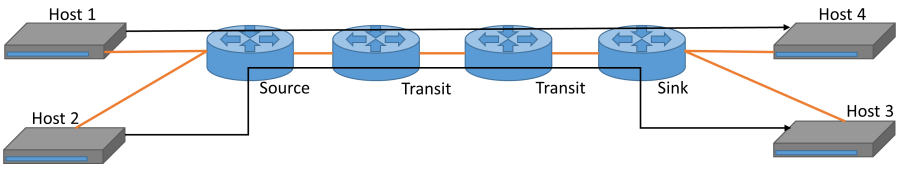


Figure 3.7: *INT benchmark network setup with four switches and two flows. Each black line represents one flow.*

3.4 data plane firewall

A proof of concept data plane firewall was designed and implemented. The idea of the firewall was to block outgoing and incoming traffic towards or from a specific network device. The firewall works by associating MAC addresses with port numbers, meaning that each switch needs a rule that matches the incoming or outgoing port with the MAC source or destination addresses respectively. These rules are implemented as table entries. If a packet does not have matching table entries that packet is dropped directly at the switch. Packets that should be dropped are marked by using the *mark_to_drop* action, which is provided by simple_switch. The firewall logic is visualized in figure 3.8. The firewall logic only resides in the ingress pipeline. After the parser the MAC source Address is matched against the table. If the incoming port is valid for that address the next step is basic forwarding with IPv4, other protocols are not supported. The resulting MAC Destination Address is then matched against the outgoing port, if it is valid the packet is sent to the traffic manager. The table entries that were needed for each switch was added with simple_switch_CLI after startup.

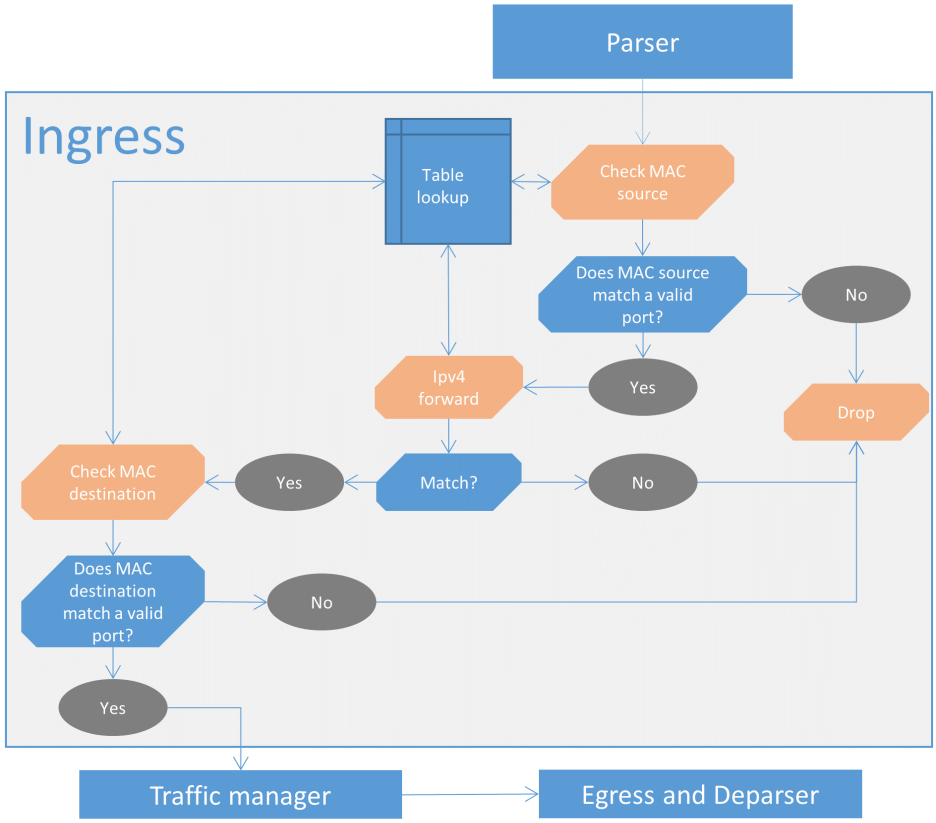


Figure 3.8: *Flowchart of the P4 firewall logic. The logic completely resides in the ingress pipeline. Rules in the table decides if a packet should be dropped or not.*

To test that the firewall worked as intended, false packets generated with Scapy [30] were injected with MAC addresses that were already present in the network. The network structure was the same as when testing Shorternet, see figure 3.2. In this case, however, Host 4 is initially not part of the internal network and should therefore be blocked by the firewall until a table entry is written which allows packets to flow to and from Host 4.

4

Results and Discussion

In this chapter, the given results from the three conducted tests are presented and discussed. Furthermore, general concerns regarding security related to P4 are discussed. Lastly, potential ideas for further development are discussed.

4.1 Shorternet

After conducting the custom-made tests to benchmark the performances of the standard and Shorternet protocol stacks it was noted that Shorternet provided an improvement in not only bandwidth, which was expected, but also in latency and jitter.

4.1.1 Effective bandwidth

The results concerning the received effective bandwidth of both tests are visualized in figure 4.1 together with a comparison between the two shown in figure 4.2. Here it is noted that the bandwidth received while using Shorternet is ever so slightly larger than for the standard solution continuously throughout the test. It is also noted that the effective bandwidth gain of using Shorternet over the standard solution is inversely proportional to the packet lengths. For small packets the effective bandwidth is almost doubled whereas for the larger packets the gain is below 5%. Theoretically, if the transmission speed and packet length is the same for both protocol stacks then the received effective bandwidth and gain should equate accordingly.

$$B_{\text{eff}} = \text{packet rate} \cdot \text{packet length} \cdot \frac{\text{payload length}}{\text{packet length}} \quad [\text{bytes/s}] \tag{4.1}$$

For Shorternet the headers account for 12 bytes of the packet length whereas for the standard solution the headers account for 42 bytes. Thus, the payload lengths are equal to packet length - 12 for Shorternet and packet length - 42 for the standard.

$$B_{\text{gain}} = \frac{B_{\text{Shorternet}}}{B_{\text{Standard}}} = \frac{\text{payload length}_{\text{Shorternet}}}{\text{payload length}_{\text{Standard}}} \tag{4.2}$$

Assuming a packet rate of 1000 pps the calculated gain in effective bandwidth for varying packet lengths is presented in table 4.1.

packet length (Bytes)	Gain (%)
74	93.75
300	11.62
850	3.713
1514	2.038
9000	0.335

Table 4.1: *Table of theoretically calculated gain in effective bandwidth for different packet lengths using a set packet rate of 1000 pps*

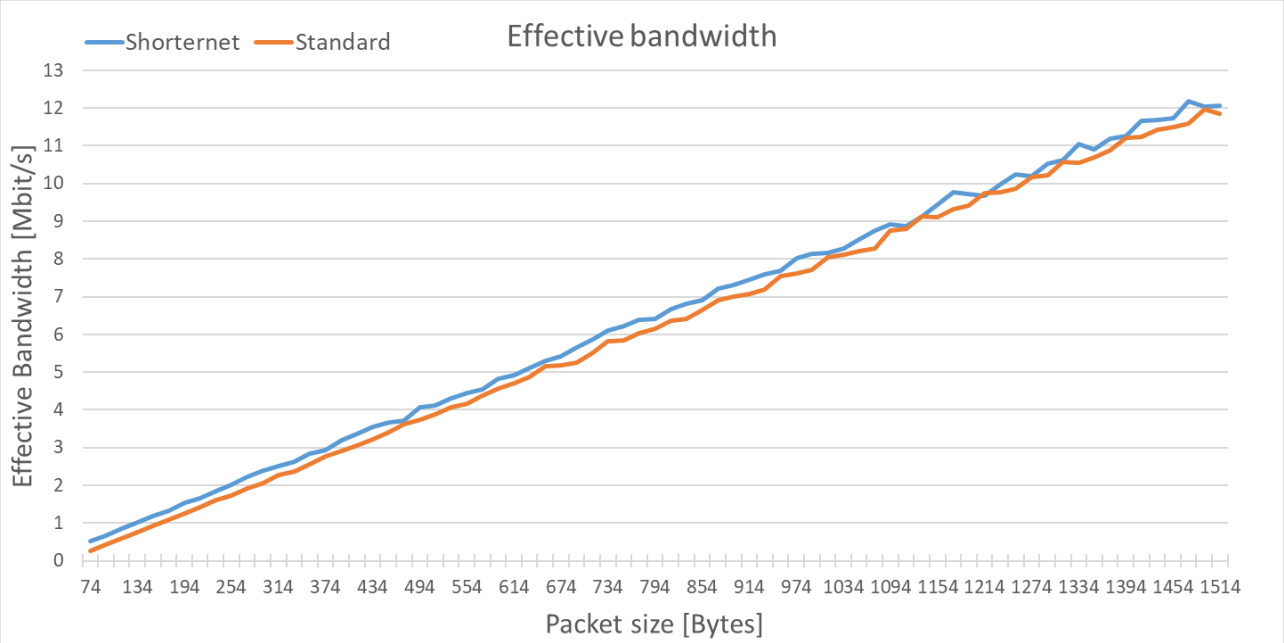


Figure 4.1: *Plot of the measured effective bandwidth as a function of packet length for Shorternet (blue) and standard TCP/IP (orange).*

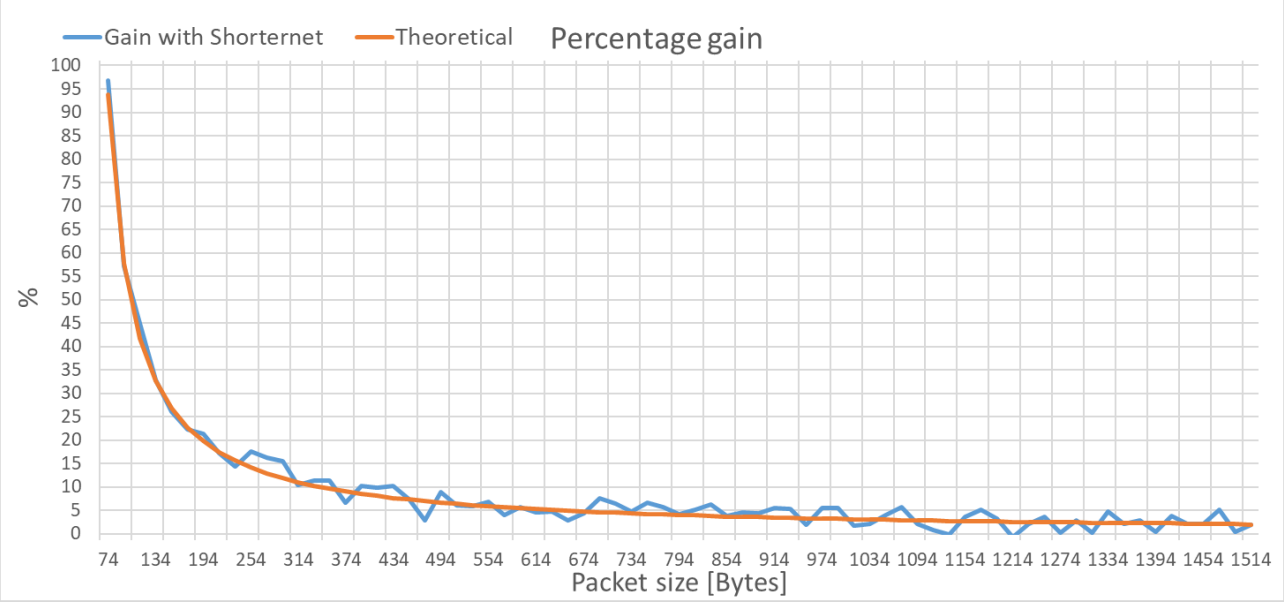


Figure 4.2: *Plot of the measured, together with the theoretically calculated, gain in effective bandwidth from using Shorternet compared to TCP/IP as a function of packet length. The blue curve represent the measured gain and the orange curve represent the theoretical gain.*

Here it is noted that for small packets, Shorternet sees a large gain in effective bandwidth whereas for larger packets this gain reduces towards zero with increasing packet lengths. This is to be expected as it is the relation between payload length and packet length that determines the gain in effective bandwidth when the packet rate is fixed according to (4.2). Since Shorternet always have 30 more bytes of payload for equal packet lengths it can be derived from (4.2) that the gain can be seen as $\frac{30+x}{x} = \frac{30}{x} + 1$ where x is the payload length of the standard solution and +1 indicate that Shorternet have a larger payload and thus a positive gain. This resulting $\frac{30}{x}$ factor is inversely proportional to the packet lengths which approaches zero for larger packet lengths. By comparing the practical test results with the theoretical, see figure 4.2, it is shown that the provided result clearly resemble the theoretical calculations. Derived from these results, both the theoretical and experimental, an improvement in effective bandwidth is clearly provided when removing redundant information within a protocol stack. However, one must consider how this relates to actual data transmissions within a network. There is an obvious potential for increased bandwidth within networks handling small packets, even an increase of 5% may prove beneficial for the commercial success of a product.

4.1.2 Latency and jitter

By analyzing the latency of the two protocol stacks, shown in figure 4.3, it can be seen that the latency of the Shorternet stack seems more stable and about 3 ms lower on average compared to the standard stack. The latency values are derived by estimating an average of the latency of every packet traversing the network during the test, as described in section 3.2.1. It was also noted that regarding packet losses, see figure 4.4, the packet loss rate was less than 0.01% during both flows, except for the standard protocol stack with packet length 694 bytes which reached about 0.025%. In comparison, the standard protocol stack experienced packet loss on more flows and also dropped a larger amount of packets than Shorternet. Since the latencies of the dropped packets are not included the amount of packets dropped reduce the sample size when estimating the average latency. Nevertheless, the number of lost packets is comparably quite few and should not affect the estimated average latency as much as seen in figure 4.3. The test scripts for both protocol stacks are equal in the way latency is derived as well as the emulated network, thus, it is reasonable to say that the improved latency comes from the reduced headers. From observations of the estimated jitter, plotted in figure 4.5 it is noted that the jitter is significantly lower for Shorternet with about a 7 ms difference on average between the two. Additionally it seems as though the jitter, similar to the latency, is not dependent on the packet lengths for both protocol stacks. As jitter is the estimated standard deviation of the latency, this suggests that Shorternet provides a more stable latency. This could also be derived from the latency results in figure 4.3. The estimated average latency of Shorternet seemed almost constant, whereas for the standard protocol stack the latency has a lot of variance, assuming that the latency is not dependent on packet length.

The throughput of an emulated Bmv2 P4 switch partly depends on the amount of match-action tables and table entries, see section 2.2.2. During the tests to compare Shorternet with standard UDP over IP the amount of match-action tables and keys were equal and the actions taken were similar. These included one table lookup and one forwarding action where one vs four lines of code were executed, which produce a minuscule difference in computing time. Therefore, the reduced latency could rather be due to that the switches parse less information. For the Shorternet protocol stack the parser no longer parse any layer three information, since it is no longer present in the packets. Another aspect that further suggests that the improvement is related to the switch processing the packets quicker for

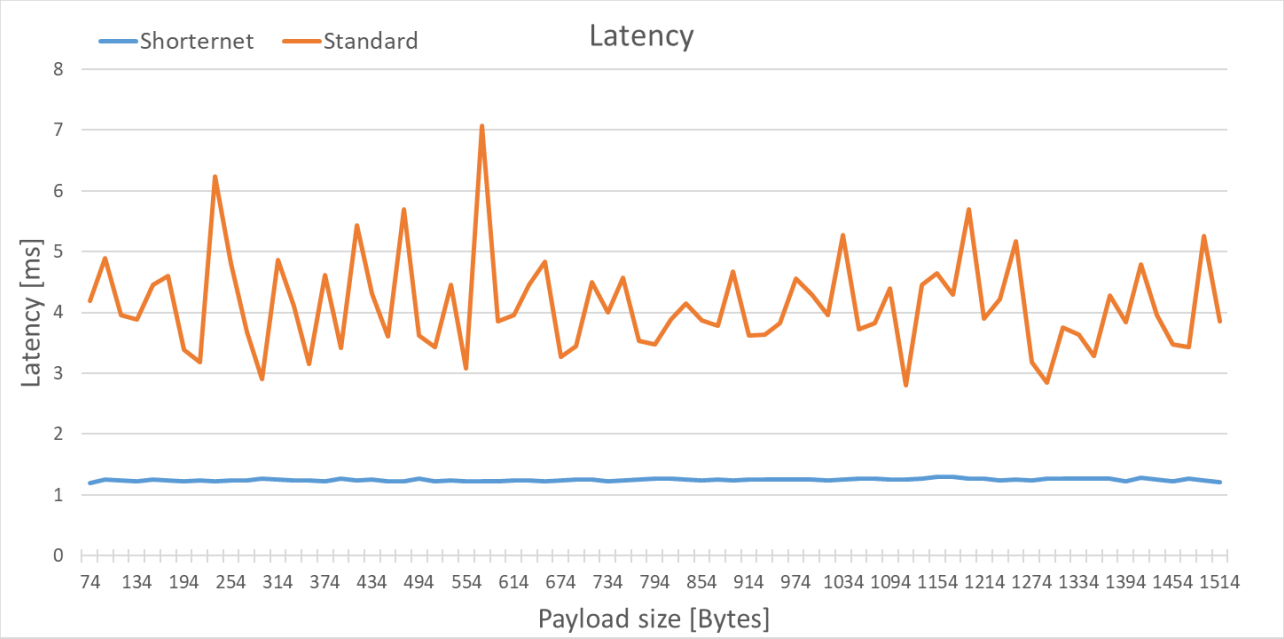


Figure 4.3: *Plot of the estimated average measured latency as a function of packet length for Shorternet (blue) and standard TCP/IP (orange).*

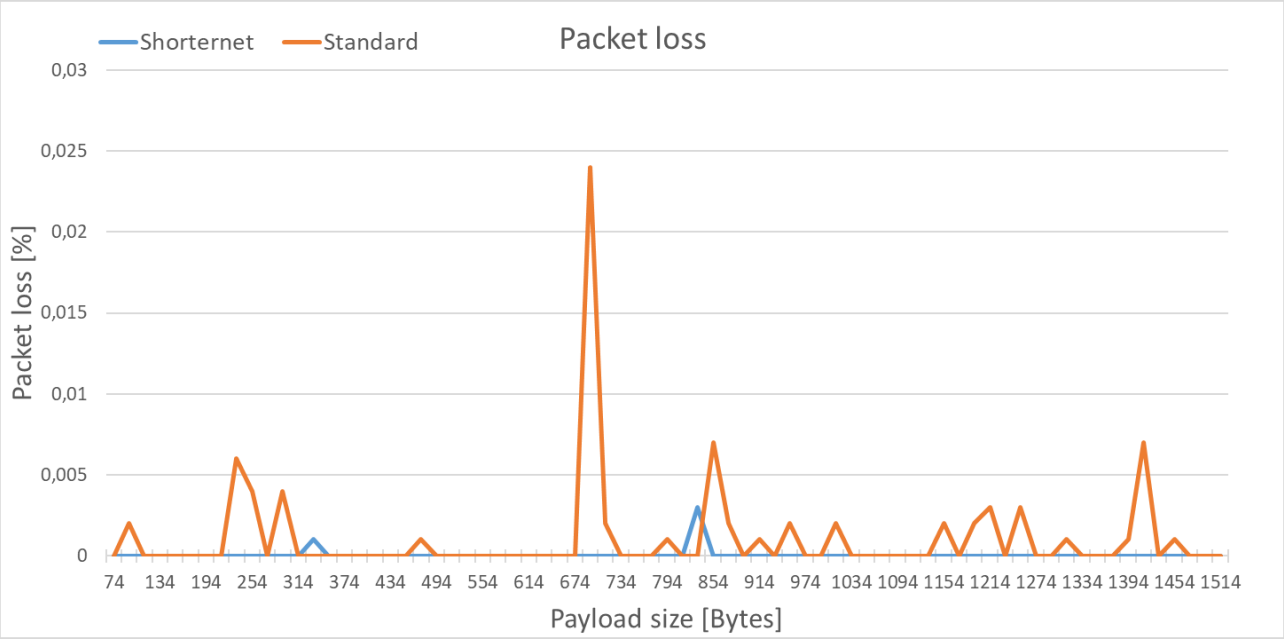


Figure 4.4: *Plot of the calculated packet loss in percent(%) when using Shorternet (blue) and standard TCP/IP (orange) for each flow of varying packet lengths.*

Shorternet is that the latency does not seem to change with increased packet lengths or when there are fewer instances where packets are dropped. Thus, the reduced latency from the tests is dependent on the fixed amount of headers that have been removed, which once again would correspond to less information processed at the parser since the raw payload is not processed at the switches.

It would have been interesting to test the effects of adding or removing more headers, should there

have been more time. A test could have been conducted with a fixed packet length but with varying amounts of header bytes to parse, while still performing the same ingress pipeline processing. This could have helped to identify more clearly what is causing the reduced latency. There is a possibility that the latency improvement of Shorternet is related to the characteristics of the emulated Bmv2 switches rather than the P4 program itself. A study [34] was made by H. Hasanin et al. where one of their tests regarding P4 was to test the cases of parsing more headers in different network hardware. In the test they go from parsing Ethernet, IPv4, UDP, and Precision Time Protocol (PTP) headers to increase the amount of headers parsed by adding dummy headers to the stack. The results from their test was that more headers to parse do provide an increase in latency in the order of single microseconds, depending on which P4 target was used. It was also noted from their results that increased packet lengths also provide an increase in latency in the order of approximately 0.1-5 microseconds, depending on which P4 target was used. Because of their results and the much lower order of which the latency increased when parsing more headers it is likely that the results from the tests performed in this project is dependent on the Bmv2 switches. With a test on actual hardware the latency improvement of using Shorternet over standard protocols may look similar to the results from the study by H. Hasanin et al.

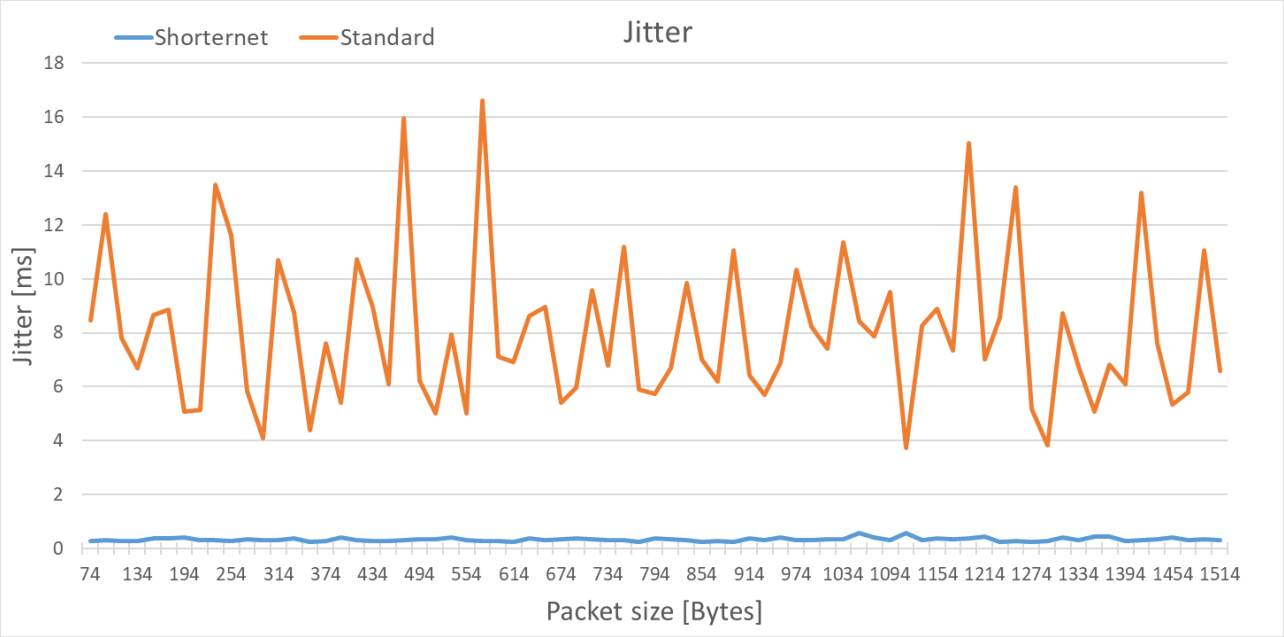


Figure 4.5: *Plot of the estimated jitter as a function of packet length for Shorternet (blue) and standard TCP/IP (orange).*

4.1.3 General discussion about Shorternet

To get a better understanding of how the gain in effective bandwidth relates to actual transmissions within a network, the following example is presented. Assume there is a large transmission of 10 GB of data. In this case, an increase of 5% in effective bandwidth equates to 500 MB of data, which is roughly estimated the same as 665000 packets of 750 bytes (disregarding headers). In certain scenarios packets are sent on-the-fly meaning that the sender does not wait for packets to be filled up to a certain point but rather spew out packets as the data arrives. For such scenarios this improvement is negligible. Regarding other scenarios this could lead to less packets being transmitted potentially

resulting in less packets lost and re-transmitted and also less packet processing at the receiver. Pair this together with the potential improvement in latency and the total transmission and processing time has been reduced by a decent amount, proving the concept to be somewhat successful.

As these tests were done in an emulated environment using a single laptop one should consider this when discussing the calculated metrics. Another aspect that should be considered was that the packet rate had to be artificially created with a sleep method in the test scripts due to the slow processing power of the switches, when emulated on a laptop. While this ensured a more fair comparison between the two protocol stacks it almost removed queue build-ups completely at the switches. The traffic behaviour might, therefore, be miss-representative of a real-world scenario.

From the presented results it seems that the Shorternet protocol stack provides the network with improvements concerning both effective bandwidth, latency, and jitter. In retrospect, a test consisting of several flows could have been more suitable to provide more representative data regarding network performance of a real-world scenario. Nonetheless, the test provided data that could be used to compare two different types of flows and what effect this has in a controlled environment. Furthermore, due to the removal of the third layer in the stack some functionality is removed which must be considered. In a traditional setup, the layer two switching is responsible for physical addressing and error correction, whereas layer three is responsible for logically routing addresses, error handling, packet sequencing etc. For instance, without layer three IP packet fragmentation is no longer available. This could potentially be modified to a layer two field instead, if one deems that functionality necessary. Moreover, the network is no longer able to utilize applications which assume that the traditional protocol stack is used, e.g internet as this depends on IP routing capabilities or *iperf* and similar tools. The question of whether or not the removal of layer three protocols is a suitable modification rather boils down to if it is needed. In this case, the modification to shorten the MAC addresses rely on the network being relatively small in size. Additionally, if the network is well-defined, IP is not needed to keep track of potential new hosts. P4 is flexible and enables the system designer to choose e.g address lengths and which protocols are needed. In this case Shorternet is more or less a proof of concept, rather than a solution for all systems. This clearly shows the utility aspect of P4 as a tool since with P4 it is more flexible to adapt and mold the network to what the system designer requires of the network.

4.2 INT

The results from the first test with INT enabled or disabled can be seen in figures 4.6, 4.7, 4.8 and 4.9. The figures show that there is almost no difference for networks with small diameter when congestion is avoided. For higher transmitted bandwidths the network experiences congestion, which is worsened with INT enabled. This is expected because the number of packets double as the sink generates a report for each packet, and the header size is increased for INT packets. There is quite a difference in the bandwidth for five switches compared to the smaller network diameters, see figure 4.6. The reason for this can be found in figure 4.8, which shows that the packet loss for five switches with INT enabled is more severe compared to the other cases. It can be noted that for five switches with INT enabled the packet loss starts to increase at a lower transmitted bandwidth. Furthermore, the average packet loss is around 0-2% until congestion occurs in the network. For latency there was a quite large impact from INT, see figure 4.7. The increased latency is a sign of congestion in the switches. The results showed that the latency is increased with both INT enabled and disabled. However, when INT

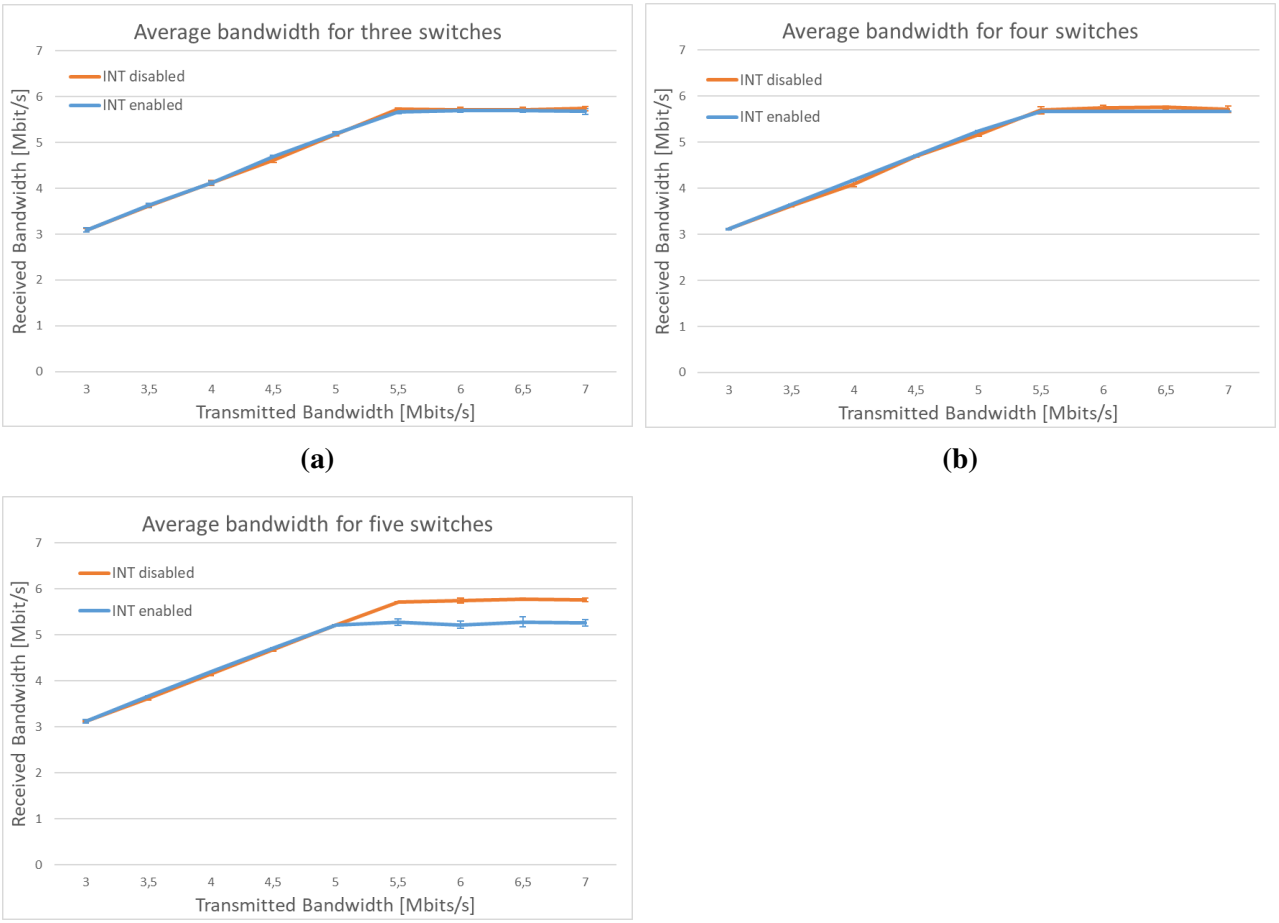


Figure 4.6: *Plots of the average measured bandwidths for a network diameter of three, four and five switches respectively. For a) and b) the results are similar but in c) the bandwidth when INT is enabled is lower compared to when INT is enabled. The error bars represent one standard deviation of the measurements.*

was enabled the congestion was more severe due to the increased amount of overhead. The network performance declined with the network diameter, which points to the added header size since the number of packets was similar for each topology. When the transmitted bandwidth reached 5 Mbit/s the bandwidth decreased for five switches, due to severe congestion. Jitter, nevertheless, saw no major difference, see figure 4.9. Moreover, there were some outliers for jitter, especially when INT was enabled for three switches. The reason why these outliers occurred was never identified and they are therefore not removed. It should be noted that the latency when INT was disabled saw no major difference with more switches added.

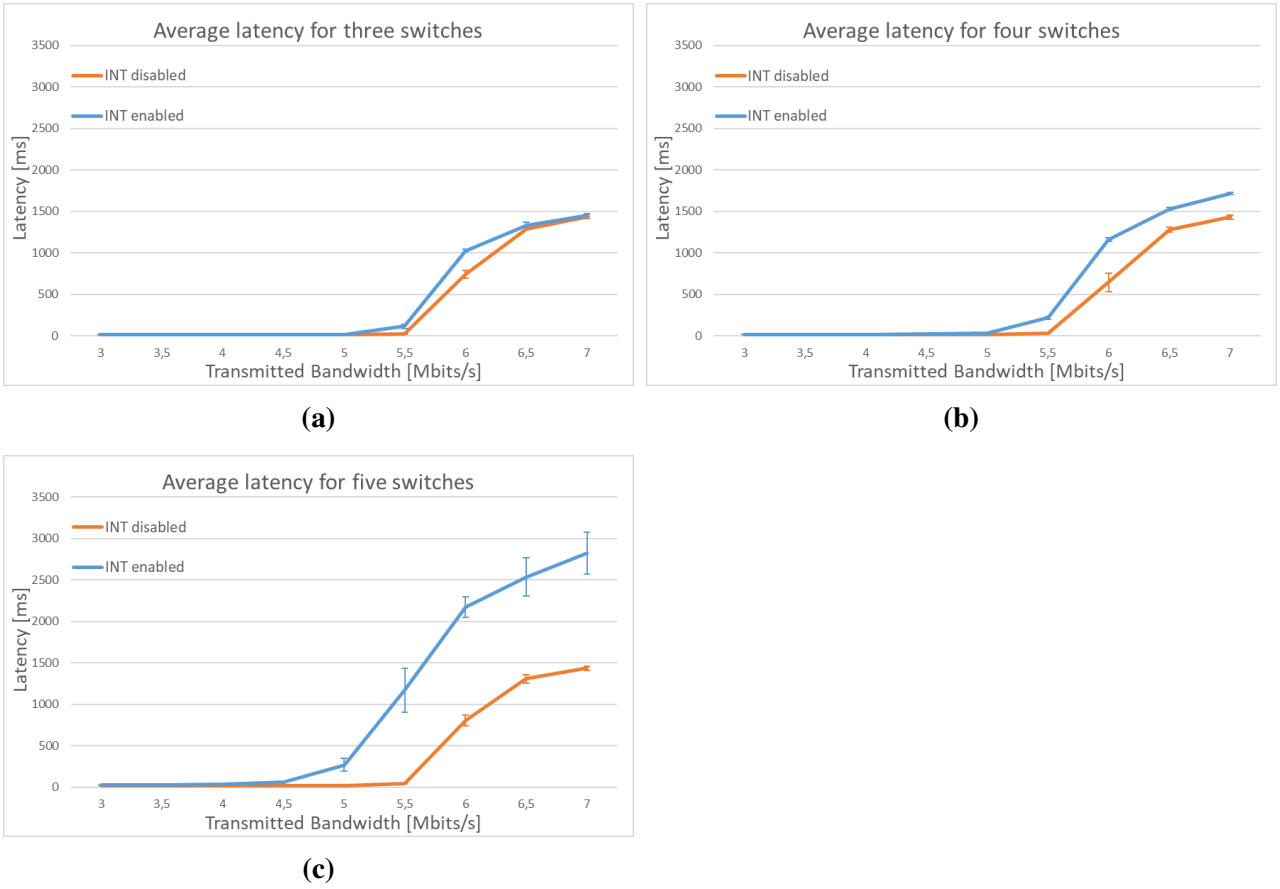


Figure 4.7: *Plots of the average measured latency for a network diameter of three, four and five switches respectively. The latency is higher when INT is enabled. Furthermore, the latency increases with the network diameter when INT is enabled. The error bars represent one standard deviation of the measurements.*

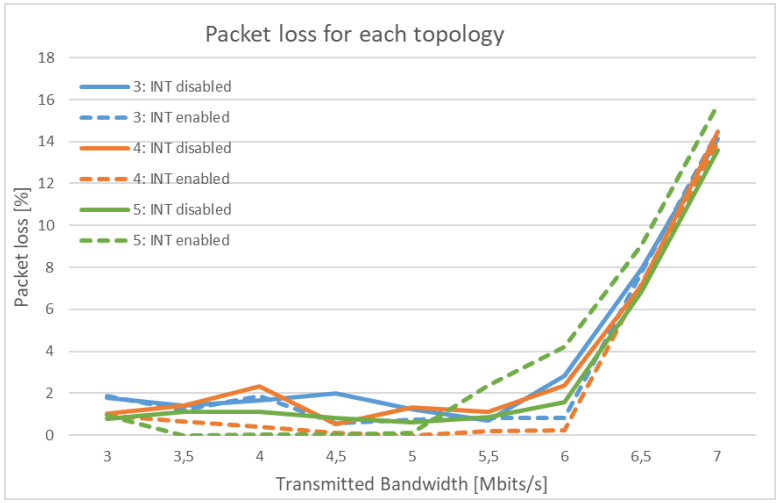


Figure 4.8: *Plot of the packet loss percentage for each network diameter with INT enabled and disabled. The packet loss is the total amount of packets dropped versus the total amount of packets sent for a transmitted bandwidth.*

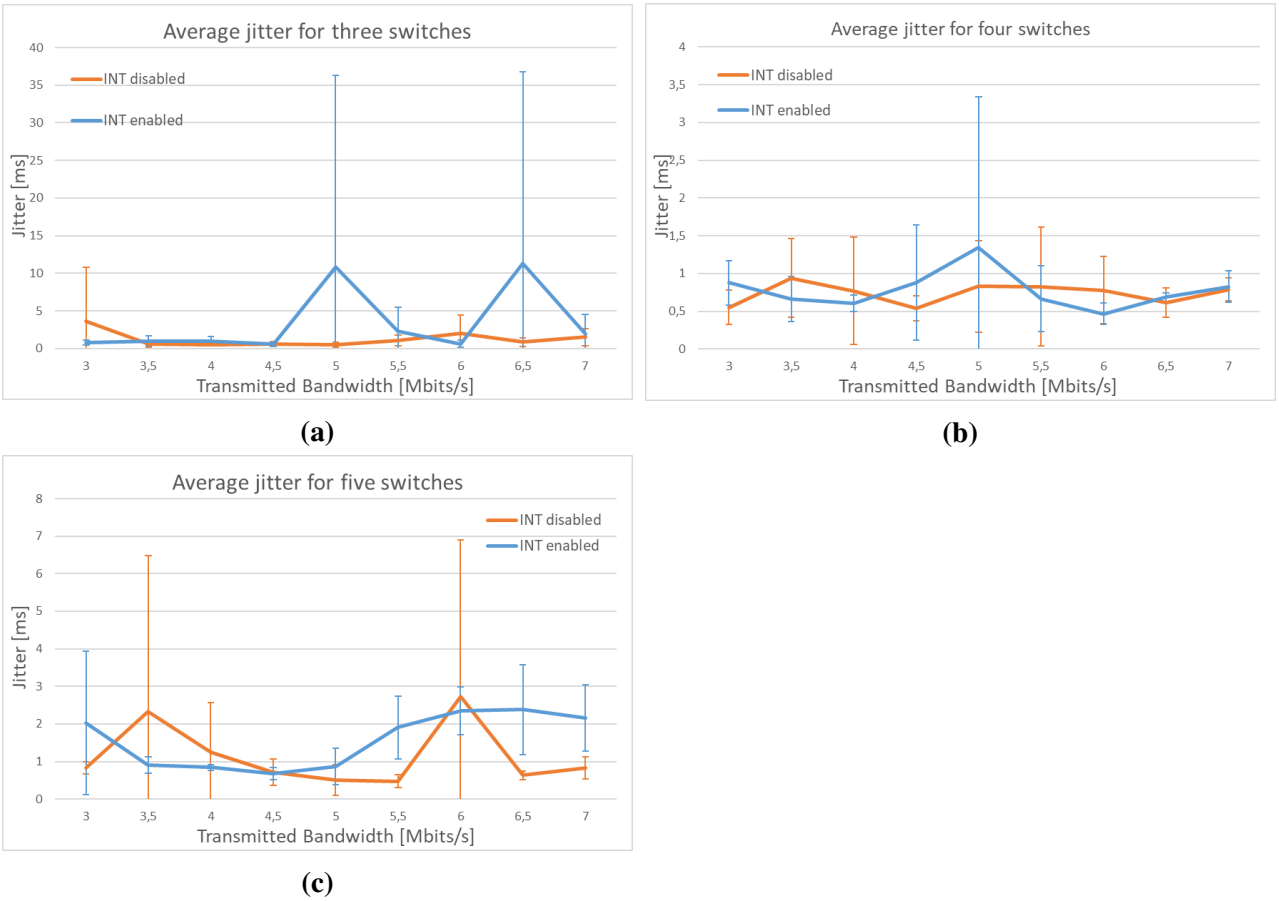


Figure 4.9: *Plots of the average measured jitter for different network diameters. The jitter is similar for all network diameters. There exists some outliers which affects the results. However, the outliers are present with INT enabled and disabled. The error bars represent one standard deviation of the measurements.*

In the second test where an additional flow was introduced, see figure 4.10, the results showed that the bandwidth, see figure 4.10a, with INT enabled and disabled were similar until the transmitted bandwidth reached 5 Mbit/s. After that point the gap increases, this is due to the same reasons as in the first test where packets were dropped due to congestion. However, a certain transmitted bandwidth should match the scenarios of the first test but with a 2 Mbit/s difference because the amount of packets is roughly the same. The reason behind the larger gap could be related to the fact that in the first test each packet was sent with a consistent frequency which is favorable in a networking environment. With two flows the frequencies between these flows are not necessarily matched, which can cause more packets to be received at a certain time frame. This will, in theory, increase the jitter as the interval between packets changes, comparing figure 4.10e with 4.10f it is apparent that the jitter increased with multiple flows. The latency of the network, see figure 4.10c, showed similar effects from enabling INT as in the first test, see figure 4.10d. However, this time the congestion started at a lower transmitted bandwidth of 3.5 Mbit/s, which is natural since the second flow of 2 Mbit/s is present. It should be noted that the performance of the network regarding bandwidth, latency, and jitter is good while congestion is avoided, for example 3-3.5 Mbit/s.

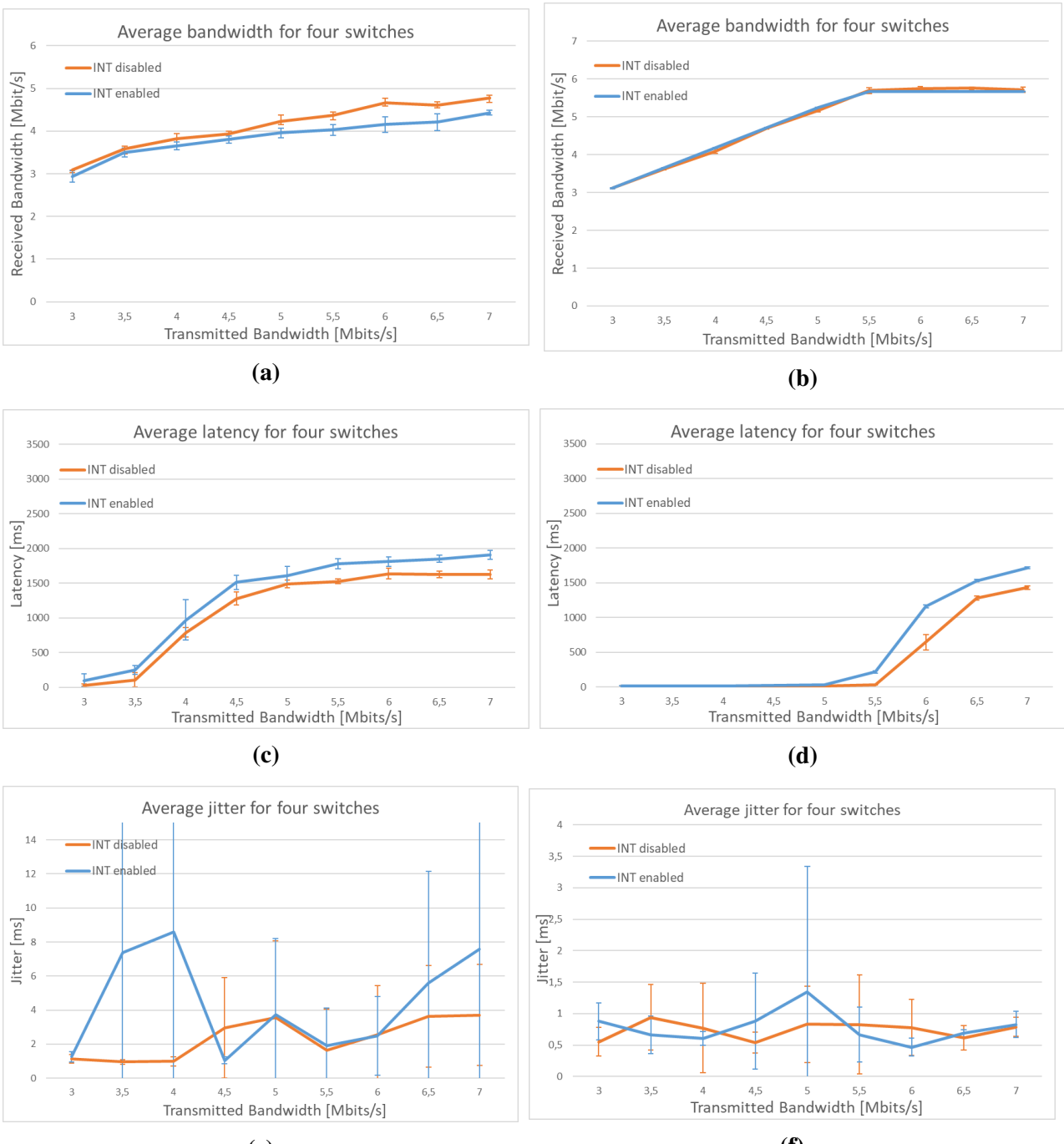


Figure 4.10: *Plots of the measured bandwidth, latency, and jitter for two flows with four switches (left) and for one flow and four switches (right). The error bars represent one standard deviation of the measurements.*

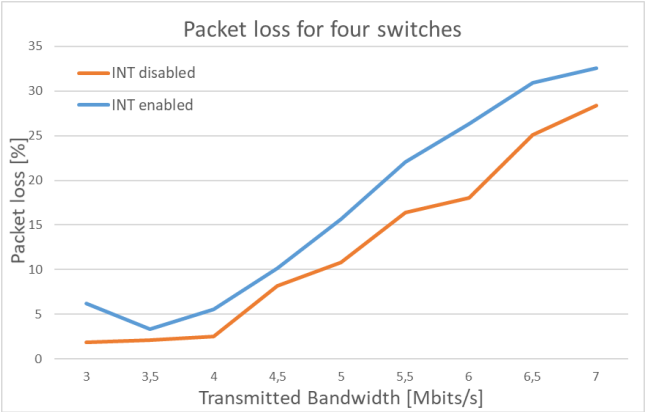


Figure 4.11: *Plot of the packet loss percentage with two flows and four switches with INT enabled and disabled. The packet loss is the total amount of packets dropped versus the total amount of packets sent for a transmitted bandwidth.*

An important aspect to consider is that the network performance is dependent on the system performance. The maximum bandwidth was limited for each switch and host to reduce the effect of this dependency. However, because each switch shared the same system resources they still affected each other to some extent. The Bmv2 target performance is also dependent on the program complexity. While there was the same amount of table entries for both tests, the P4 program with INT disabled is much less complex compared to the P4 program with INT enabled. The performance of a hardware switch might not be affected by the program itself as it was for the Bmv2 switch. Therefore, multiple reasons points to do the same tests on hardware switches to get the full picture.

The INT protocol that was implemented is quite simple in design. If legacy switches were present in the network there might be complications with the INT headers. Additionally, in this project, only UDP traffic was exposed to INT. Because the programmer has full control, INT can be added to any other protocol in order for flows other than UDP traffic to be able to obtain vital network information. Also INT was applied to every single UDP packet, it might be interesting to investigate if that is needed and if it is possible to lower the amount of INT packets that circulate the network to reduce queue build-up. For example adding INT at every other packet would reduce the number of packets with added headers by half. Also, the amount of reports would be halved as well.

The results from both tests, show that solely enabling INT has a negative impact on the network. The idea was to use INT to deliver fine-grained metrics to a CC algorithm which would avoid the queue-build up that occurred during the tests. For applications where latency is crucial, CC is a way to keep the queue build-ups at a minimum. Low-latency does not come without a cost, in this case the trade-off is lower bandwidth. As mentioned in section 2.2.3c, HPCC caused almost zero-queues but had a 5% loss in bandwidth. As a result HPCC achieved 95% faster flow-completion time compared to earlier implementations. HPCC uses 2 bytes for header and adds 8 bytes of data at each hop. This is one byte less INT-Header and 4 bytes less INT-Data compared to the implementation in this project. The header sizes depend on what metrics that are gathered and what resolution they have. R. B. Basat, et al. proposed PINT [35] which lowers the INT header overhead to as low as two bytes per packet, which was evaluated together with HPCC. In this project the header sizes were not optimized, which leaves room for improvements.

There are multiple use cases beside CC for INT. Load balancing and anomaly detection were two other examples mentioned in section 2.2.3c. Load balancing is similar to CC but instead a flow can be separated into multiple flows distributed between switches or hosts rather than limiting the bandwidth. INT brings higher resolution in metrics such as load of different switches and hosts, which can provide better decisions of where traffic should be forwarded. Anomaly detection can be used to find patterns of attacks such as DDoS and network breaches. The telemetry data could be provided to a controller which can take appropriate action to stop or mitigate these attacks. Furthermore, accountability could be provided if data from INT is saved, which can be used at later stages in investigations of attacks such as breaches. Data from INT could also be used to optimize networks because it is possible to pin-point where there might be bottlenecks or security risks. However, INT produces a lot of data and it might not be feasible to store such large amounts of data. To summarize, there are many applications which could be applied with the use of INT. Depending on what application is to be implemented, it is up to the developers to investigate whether the benefits outweigh the added overhead or not.

4.3 General security aspects

In this section some concerns regarding security flaws and improvements are presented and discussed for programmable data planes with P4. These include bugs, debugging of P4 code, and the idea of added cryptography at the Link layer level.

4.3.1 Security concerns and bugs

Agape et al. investigates in their position paper [36] what new security challenges P4 introduces. They point out that security studies regarding OpenFlow still apply, for more information regarding OpenFlow, see [37]. Agape et al. explain a number of security flaws. For example, switches that run P4Runtime are vulnerable to Man-in-the-middle attacks and channel flooding. In the case of Man-in-the-middle-attacks, important non-encrypted gRPC messages which contain information such as configuration files, tables and control instructions can be captured between the switch and controller. This information can be used to spoof the controller. In the case of channel flooding the P4Runtime agent on the controller or the switches can flood each other, resulting in slow response time or denial of service of the controller or higher latency of the network.

P4 enables the programmer to define the data plane completely, and with faster development, the risk for runtime bugs is inherently increased. This might not be the case for certain programming languages which includes necessary checks to find bugs. However, for P4 this is not the case and additionally the network developer needs think how the P4 logic handles an unexpected value or protocol. Otherwise bugs and exploits might be possible as a result. M. Dumitru, D. Dumitrescu and C. Raiciu investigates how bugs in P4 programs can be exploited in different targets [38]. The authors point out that P4 suffers similar weak-points as in the programming language C. They further explain that there are some apparent differences that results in weaker attacks towards P4 compared to C. These are as follows:

- In P4, if a read or write operation is invalid when running the program then only the value (reads) or the memory location (writes) of that operation are invalid. In C if these faults occur, then the whole behaviour of the program is undefined after that point resulting in an execution error.

- Code injections that are present in other languages are not possible in P4 since the code is immutable after deployment.
- In P4 the execution order is immutable, which provides control-flow integrity [39] by default.

They uncover in testing that the Bmv2 switch leaks information from prior packets. Regardless, Bmv2 is not meant as a production target [13], which the authors also note.

It is clear that with expanded programmability, there is a greater risk for bugs slipping through to production networks. These bugs could impose security flaws, fault and downtime. As a result multiple debugging methods and tools have been proposed [40], [41], [42], [43]. ASSERT-P4 [42] proposed by L. Freire et al. and Vera [41] proposed by R. Stoenescu both use symbolic execution to find bugs like, parsing/deparsing errors, loops, etc. Both work by representing the P4 program in a different form which enables them to use an engine to find bugs. The overall functionality of the engines are that the different parts of a P4 program are represented and then different packets are injected. P4Tester [40] proposed by Y. Zhou et al. finds bugs by injecting certain probes in a network. These probes use different headers (Ethernet, IPv4, and so fourth) to invoke table rules and then save information in the same packet that can be used for debugging. Y. Zhou et al. proposed P4DB [43] which lets the programmer add "debugging snippets" in the code while developing. These snippets are similar to standard breakpoints. Breakpoints enable the programmer to pause code during runtime and inspect variables. With P4DB the execution is not paused but rather the programmer gets the state of different elements at the points of interest instead.

4.3.2 Cryptographic Hash

By providing network security at lower levels such as the Link layer the network can be safeguarded against cyberattacks, e.g Man-in-the-middle, eavesdropping, etc. which often exploit the vulnerability of the link layer. One way of providing such security is through using MACsec (802.1AE-2006), which is a security protocol introduced by the IEEE 802.1 work group 2006 [44]. MACsec uses encryption to ensure confidentiality, integrity and authenticity between peers in the network. F. Hauser et al. proposes P4-MACsec which implements MACsec in a SDN with P4 switches [45]. The authors point out that automated deployment of MACsec in legacy switches is not feasible. P4-MACsec leverages a SDN controller for automated deployment. Furthermore, P4 is used to implement MACsec functionality in the data plane. They further point out that they used the Advanced Encryption Standard in Galois/Counter mode (AES-GCM) for encryption and decryption. However, cryptographic hashes are not natively supported by P4 targets, at the current time. D. Scholz et al. investigate the feasibility of implementing cryptographic hash algorithms in different P4 targets such as CPUs, NPUs and FPGAs [46]. They concluded that no hash algorithm, that they tested, deliver enough performance on any platform. They recommend that P4 targets should implement a family of cryptographic hash functions, recommended by the P4 specification, that suits the target. Because of P4 the potential of new network applications is demonstrated, which can motivate the switch manufactures to develop and implement new features.

4.4 Firewall

During the test, the firewall blocked packets with spoofed MAC addresses and Host 4 had no access to the network. However, the switch and controller (simple_switch_CLI) runs P4Runtime, which is vulnerable to man-in-the-middle attacks, see 4.3.1. If an intruder gains access to the controller, new table entries that gives access to the network can be installed. Also, if the attacker has physical access to the switch, entry to the network is not a difficult task.

For a complete solution, controller logic needs to be implemented together with the switch P4 behavior. This is currently not implemented. How hosts/network devices should be verified and how these rules are distributed to the networks switches was not investigated. Also any impact on performance was not measured. Currently, this implementation of a data plane firewall adds two rules for each individual host and switch. One rule for outgoing traffic and one rule for incoming traffic. The number of table entries should affect performance of the Bmv2 switch according to previously mentioned factors described in section 2.2.2. However, a production switch might not have the same limitations as the Bmv2.

Currently, this firewall implementation only checks the MAC addresses and port numbers. However, other rules could be implemented, for example the switches could check the destination port of the transportation layer headers and drop packets accordingly. There is also potential for other ideas of rules. One such rule could be that a new host may only initiate communication between already present hosts if these have previously initiated communications with the new host beforehand.

4.5 Future work

This thesis implemented prototypes of different applications for P4 with a more "proof of concept" methodology rather than actual implementations. These prototypes saw potential and could be expanded upon and also include tests using real hardware and not only through emulated systems. Performance metrics from tests on real hardware could bring more insight into how these protocols impact the network. Some ideas on how these concepts can be built upon, which would have been interesting to evaluate and analyze, are presented.

The idea of Shorternet was to conceptualize custom-made protocol stacks that differ quite a lot from the standardized model. However, as was discussed, this removes some functionality. An interesting idea for further study could be whether or not a hybrid network is feasible, which may utilize both completely custom-made models as well as the standards. This may prove useful for networks with a well defined and fixed local network structure but has connection points outwards to either WAN or another distant LAN. Another aspect where a hybrid network could prove useful is to be able to combine tailor-made protocols while still retaining the standard functionality for applications that expect the standard protocols. Furthermore, Shorternet showed the flexibility of P4 and its easy to use concept for packet processing. This brings possibilities for new protocols that can expand upon current standardized protocols such as adding packet segmentation or sequencing packets in lower layers than previous to remove redundant information while retaining functionality.

INT as a tool was implemented, many applications can be built using the data INT offers. For example congestion control and load balancing. The INT protocol could, however, be optimized in multiple

ways, for example shorter headers and less INT packets in a network. It is also worth pointing out that INT can be applied to other protocols, for example TCP, if there is a need for feedback and control of these protocols.

As technology continues to evolve together with digitization, cyber-security is becoming more crucial than ever for companies to survive. With the concept of data plane firewalls there is potential for increased levels of security within computer networks. In this project one such firewall concept was presented, a simple yet effective concept. However, it saw a need for a controller to be constructed in order to apply the firewall in networks. This could be expanded upon and also include a way of dynamically adding and removing further sets of rules as well as exploring new ideas for rules.

5

Conclusion

To summarize, three different applications of P4 were implemented and investigated. Shorternet showed positive gains in bandwidth, latency, and jitter. Regarding bandwidth the gain was significant for small payloads but for larger payloads this gain was reduced as the gain is inversely proportional to packet lengths. The latency and jitter saw improvements of approximately 75% less latency and 86% less jitter on average using Shorternet, but, may have been affected by the characteristics of the emulated network. Although there were improvements regarding network performance, there were some downsides as well that are worth considering. Applications in the network that take for granted that standard communication protocols are present will not work as intended. This calls for completely custom-made solutions involving the full protocol stack, which might not be feasible. For example, a new benchmarking tool needed to be constructed because legacy tools like *iperf2* could not be used. In-Band Network Telemetry (INT) showed great potential and together with congestion control or other applications, such as anomaly detection, could provide low-latency or more secure networks at the expense of reduced bandwidth. Results showed that the added overhead does decrease the performance as more traffic and/or larger data packets are introduced to the network. But this decrease may prove to not be of significance if the INT data can be utilized with more advanced CC algorithms or other implementations. Lastly, a data plane firewall which matched physical port numbers with MAC addresses was implemented. The firewall proved to be effective and could easily be expanded upon with new rules. However, a controller that is able to verify and securely distribute rules to the switches needs to be added before the firewall is deployed in real networks. The firewall is not completely foolproof, if an attacker is able to spoof controller messages or can access the controller, this could compromise the firewall.

Programmable data planes open up a new front of security threats. P4 as a tool provides programmers with more flexibility, paired together with faster innovation cycles this may introduce more bugs into production. Because of this, several debuggers have been developed, where P4DB is one such example which helps the developer to find runtime bugs. Some existing security flaws were presented and because of the coexistence of Software Defined Networks (SDN) and programmable data planes, security concerns from SDN carry over as well.

During the project, P4 was found to be a powerful and easy to use tool. Prototypes were fast to develop and implement. With the help of the behavioral model version 2 (Bmv2) and Mininet, prototypes could be implemented and tested in an emulated environment which accelerated development, albeit not fully representing a production network. The network could be customised at a very low level, which opens up for customization and flexibility. Many applications which utilize P4 are emerging, which point to scientific interest in the language as well as practicality.

References

[1] P4 Consortium, “P4 Language and Related Specifications.” 2021. [Online]. Available: https://p4.org/specs/(accessed on: 2021-05-14).

[2] *Requirements for Internet Hosts - Communication Layers*, RFC 1122, R. Braden, Internet Engineering Task Force, Los Angeles, USA, 1989. Available: http://www.rfc-editor.org/rfc/rfc1122.txt.

[3] *Transmission Control Protocol*, RFC 793, J. Postel, Information Sciences Institute, Los Angeles, USA, 1981. Available: https://datatracker.ietf.org/doc/html/rfc793.

[4] *User Datagram Protocol*, RFC 768, J. Postel, Information Sciences Institute, 1980. Available: https://datatracker.ietf.org/doc/html/rfc768.

[5] *Internet Protocol*, RFC 791, J. Postel, Information Sciences Institute, Los Angeles, USA, 1980. Available: https://datatracker.ietf.org/doc/html/rfc791.

[6] *Internet Protocol, Version 6 (IPv6) Specification*, RFC 8200, S. Deering, R. Hinden, Internet Engineering Task Force, 2017. Available: https://datatracker.ietf.org/doc/html/rfc8200.

[7] *IEEE Standard for Ethernet*, IEEE 802.3-2018, IEEE Computer Society, New York, USA, 2018. Available: https://ieeexplore.ieee.org/document/8457469.

[8] Internet Assigned Numbers Authority (IANA), "Protocol Numbers," 2021. [Online]. Available: https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml (accessed on 2021-02-17).

[9] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, “P4: Programming protocol-independent packet processors,” *SIGCOMM Comput. Commun. Rev.*, vol. 44, p. 87–95, July 2014. [Online]. Available: https://dl.acm.org/doi/10.1145/2656877.2656890, Accessed on: 2021-01-26).

[10] M. Budiu and C. Dodd, “The P416 Programming Language,” *SIGOPS Oper. Syst. Rev.*, vol. 51, p. 5–14, Sept. 2017. [Online]. Available: https://dl.acm.org/doi/10.1145/3139645.3139648, Accessed on: 2021-01-26.

[11] P4 Language Consortium. "01 Introduction to Data Plane Programming (Stephen Ibanez)," *Youtube*, Nov. 30, 2017. [Video file]. Available: https://www.youtube.com/watch?v=qxT7DKOIk7Q (accessed on 2021-05-05).

[12] The P4.org Architecture Working Group, "$P4_{16}$ Portable Switch Architecture (PSA)," 2018. [Online]. Available: https://p4lang.github.io/p4-spec/docs/PSA-v1.1.0.pdf (accessed on 2021-02-03).

[13] Behavioral model, Version 1.14.0, [Software], P4 Language Consortium, 2021. Available: https://github.com/p4lang/behavioral-model (accessed on: 2021-05-14).

[14] Mininet, Version 2.3.0, [Software], 2021. Available: https://github.com/mininet/mininet (accessed on: 2021-02-14).

[15] A. Bas, "Performance of bmv2." 2019. [Online]. Available: https://github.com/p4lang/behavioral-model/blob/main/docs/performance.md (accessed on: 2021-05-05).

[16] A. Bas, V. Kumar, H. Hu, C.W. Cen, "P4 forwarding large packets," 2018. [Online]. Available: https://github.com/p4lang/behavioral-model/issues/567 (accessed on: 2021-03-18).

[17] The P4.org Applications Working Group, "In-band Network Telemetry (INT) Dataplane Specification." 2020. [Online]. Available: https://raw.githubusercontent.com/p4lang/p4-applications/master/docs/INT_v2_1.pdf (accessed on: 2021-02-07).

[18] The P4.org Applications Working Group, "Telemetry Report Format Specification." 2020. [Online]. Available: https://github.com/p4lang/p4-applications/blob/master/docs/telemetry_report_v2_0.pdf (accessed on: 2021-02-07).

[19] V. Jacobson, "Congestion avoidance and control," *SIGCOMM Comput. Commun. Rev.*, vol. 18, p. 314–329, Aug. 1988. [Online]. Available: https://dl.acm.org/doi/10.1145/52324.52356, Accessed on: 2021-05-17.

[20] Intel Corporation, "Intel Deep Insight Network Analytic Software." 2020. [Online]. Available: https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/network-analytics/deep-insight.html (accessed on: 2021-05-12).

[21] Y. Li, R. Miao, H. H. Liu, Y. Zhuang, F. Feng, L. Tang, Z. Cao, M. Zhang, F. Kelly, M. Alizadeh, and M. Yu, "HPCC: High Precision Congestion Control," in *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, (New York, NY, USA), p. 44–58, Association for Computing Machinery, 2019. [Online]. Available: https://dl.acm.org/doi/10.1145/3341302.3342085, Accessed on: 2021-03-11.

[22] Y. Zhu, H. Eran, D. Firestone, C. Guo, M. Lipshteyn, Y. Liron, J. Padhye, S. Raindel, M. H. Yahia, and M. Zhang, "Congestion Control for Large-Scale RDMA Deployments," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, (New York, NY, USA), p. 523–536, Association for Computing Machinery, 2015. [Online]. Available: https://dl.acm.org/doi/10.1145/

2829988.2787484, Accessed on: 2021-05-31.

[23] R. Mittal, V. T. Lam, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, and D. Zats, "TIMELY: RTT-Based Congestion Control for the Datacenter," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, (New York, NY, USA), p. 537–550, Association for Computing Machinery, 2015. [Online]. Available: https://dl.acm.org/doi/10.1145/2785956.2787510, Accessed on: 2021-05-31.

[24] J. Lim, S. Nam, J.-H. Yoo, and J. W.-K. Hong, "Best nexthop Load Balancing Algorithm with Inband network telemetry," in *2020 16th International Conference on Network and Service Management (CNSM)*, pp. 1–7, 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9269053, Accessed on 2021-05-18.

[25] M. Chiesa, G. Kindler, and M. Schapira, "Traffic Engineering With Equal-Cost-MultiPath: An Algorithmic Perspective," *IEEE/ACM Transactions on Networking*, vol. 25, no. 2, pp. 779–792, 2017. [Online]. Available: https://ieeexplore.ieee.org/document/7588075, Accessed on: 2021-05-18.

[26] J. Hyun, N. Van Tu, and J. W.-K. Hong, "Towards knowledge-defined networking using in-band network telemetry," in *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–7, 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8406169, Accessed on: 2021-05-18.

[27] J. Deng, H. Hu, H. Li, Z. Pan, K.-C. Wang, G.-J. Ahn, J. Bi, and Y. Park, "VNGuard: An NFV/SDN combination framework for provisioning and managing virtual firewalls," in *2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*, pp. 107–114, 2015. [Online]. Available: https://ieeexplore.ieee.org/document/7387414, Accessed on: 2021-05-16.

[28] H. Hu, W. Han, G.-J. Ahn, and Z. Zhao, "Flowguard: Building robust firewalls for software-defined networks," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, (New York, NY, USA), p. 97–102, Association for Computing Machinery, 2014. [Online]. Available: https://dl.acm.org/doi/10.1145/2620728.2620749, Accessed on: 2021-05-16.

[29] R. Datta, S. Choi, A. Chowdhary, and Y. Park, "P4guard: Designing p4 based firewall," in *MILCOM 2018 - 2018 IEEE Military Communications Conference (MILCOM)*, pp. 1–6, 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8599726, Accessed on: 2021-05-15.

[30] P. Biondi and the Scapy community, "Scapy," 2021. [Online]. Available: https://scapy.net/ (accessed on 2021-04-21).

[31] "iPerf 2 user documentation". [Online]. Available: https://iperf.fr/iperf-doc.php#doc (accessed on: 2021-06-07).

[32] *RTP: A Transport Protocol for Real-Time Applications*, RFC 3550, H. Schulzrinne, S. Casner,

R. Frederick, V. Jacobson, 2003. Available: https://datatracker.ietf.org/doc/html/rfc3550.

[33] "ping(8) - Linux man page". [Online]. Available: https://linux.die.net/man/8/ping (accessed on: 2021-06-07).

[34] H. Harkous, M. Jarschel, M. He, R. Pries, and W. Kellerer, "P8: P4 with Predictable Packet Processing Performance," *IEEE Transactions on Network and Service Management*, pp. 1–1, 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9220822, Accessed on: 2021-06-16.

[35] R. Ben Basat, S. Ramanathan, Y. Li, G. Antichi, M. Yu, and M. Mitzenmacher, "Pint: Probabilistic in-band network telemetry," in *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '20, (New York, NY, USA), p. 662–680, Association for Computing Machinery, 2020. [Online]. Available: https://dl.acm.org/doi/10.1145/3387514.3405894, Accessed on: 2021-06-01.

[36] A. Agape, M. C. Danceanu, R. R. Hansen, and S. Schmid, "Charting the security landscape of programmable dataplanes," unpublished. 2018. [Online]. Avaiable: https://arxiv.org/abs/1807.00128, Accessed on: 2021-05-13.

[37] R. Klöti, V. Kotronis, and P. Smith, "Openflow: A security analysis," in *2013 21st IEEE International Conference on Network Protocols (ICNP)*, pp. 1–6, 2013. [Online]. Available: https://ieeexplore.ieee.org/document/6733671, Accessed on: 2021-05-10.

[38] M. V. Dumitru, D. Dumitrescu, and C. Raiciu, "Can we exploit buggy p4 programs?," in *Proceedings of the Symposium on SDN Research*, SOSR '20, (New York, NY, USA), p. 62–68, Association for Computing Machinery, 2020. [Online]. Available: https://dl.acm.org/doi/10.1145/3373360.3380836, Accessed on: 2021-05-10.

[39] M. Abadi, M. Budiu, U. Erlingsson, and J. Ligatti, "Control-flow integrity," in *Proceedings of the 12th ACM Conference on Computer and Communications Security*, CCS '05, (New York, NY, USA), p. 340–353, Association for Computing Machinery, 2005. [Online]. Available: https://dl.acm.org/doi/10.1145/1102120.1102165, Accessed on: 2021-05-14.

[40] Y. Zhou, J. Bi, Y. Lin, Y. Wang, D. Zhang, Z. Xi, J. Cao, and C. Sun, "P4tester: Efficient runtime rule fault detection for programmable data planes," in *Proceedings of the International Symposium on Quality of Service*, IWQoS '19, (New York, NY, USA), Association for Computing Machinery, 2019. [Online]. Available: https://dl.acm.org/doi/10.1145/3326285.3329040, Accessed on: 2021-05-18.

[41] R. Stoenescu, D. Dumitrescu, M. Popovici, L. Negreanu, and C. Raiciu, "Debugging p4 programs with vera," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, (New York, NY, USA), p. 518–532, Association for Computing Machinery, 2018. [Online]. Available: https://dl.acm.org/doi/10.1145/3230543.3230548, Accessed on: 2021-05-18.

[42] L. Freire, M. Neves, L. Leal, K. Levchenko, A. Schaeffer-Filho, and M. Barcellos, "Uncovering bugs in p4 programs with assertion-based verification," in *Proceedings of the Symposium on SDN Research*, SOSR '18, (New York, NY, USA), Association for Computing Machinery, 2018. [Online]. Available: https://dl.acm.org/doi/10.1145/3185467.3185499, Accessed on: 2021-05-18.

[43] Y. Zhou, J. Bi, C. Zhang, B. Liu, Z. Li, Y. Wang, and M. Yu, "P4db: On-the-fly debugging for programmable data planes," *IEEE/ACM Trans. Netw.*, vol. 27, p. 1714–1727, Aug. 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8771219, Accessed on: 2021-05-15.

[44] *IEEE Standard for Local and Metropolitan Area Networks - Media Access Control (MAC) Security*, 802.1AE-2018, IEEE Computer Society, New York, USA, 2018. Available: https://ieeexplore.ieee.org/document/8585421.

[45] F. Hauser, M. Schmidt, M. Häberle, and M. Menth, "P4-MACsec: Dynamic Topology Monitoring and Data Layer Protection With MACsec in P4-Based SDN," *IEEE Access*, vol. 8, pp. 58845–58858, 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9044731, Accessed on: 2021-04-16.

[46] D. Scholz, A. Oeldemann, F. Geyer, S. Gallenmüller, H. Stubbe, T. Wild, A. Herkersdorf, and G. Carle, "Cryptographic Hashing in P4 Data Planes," in *2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, pp. 1–6, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8901886, Accessed on: 2021-04-13.