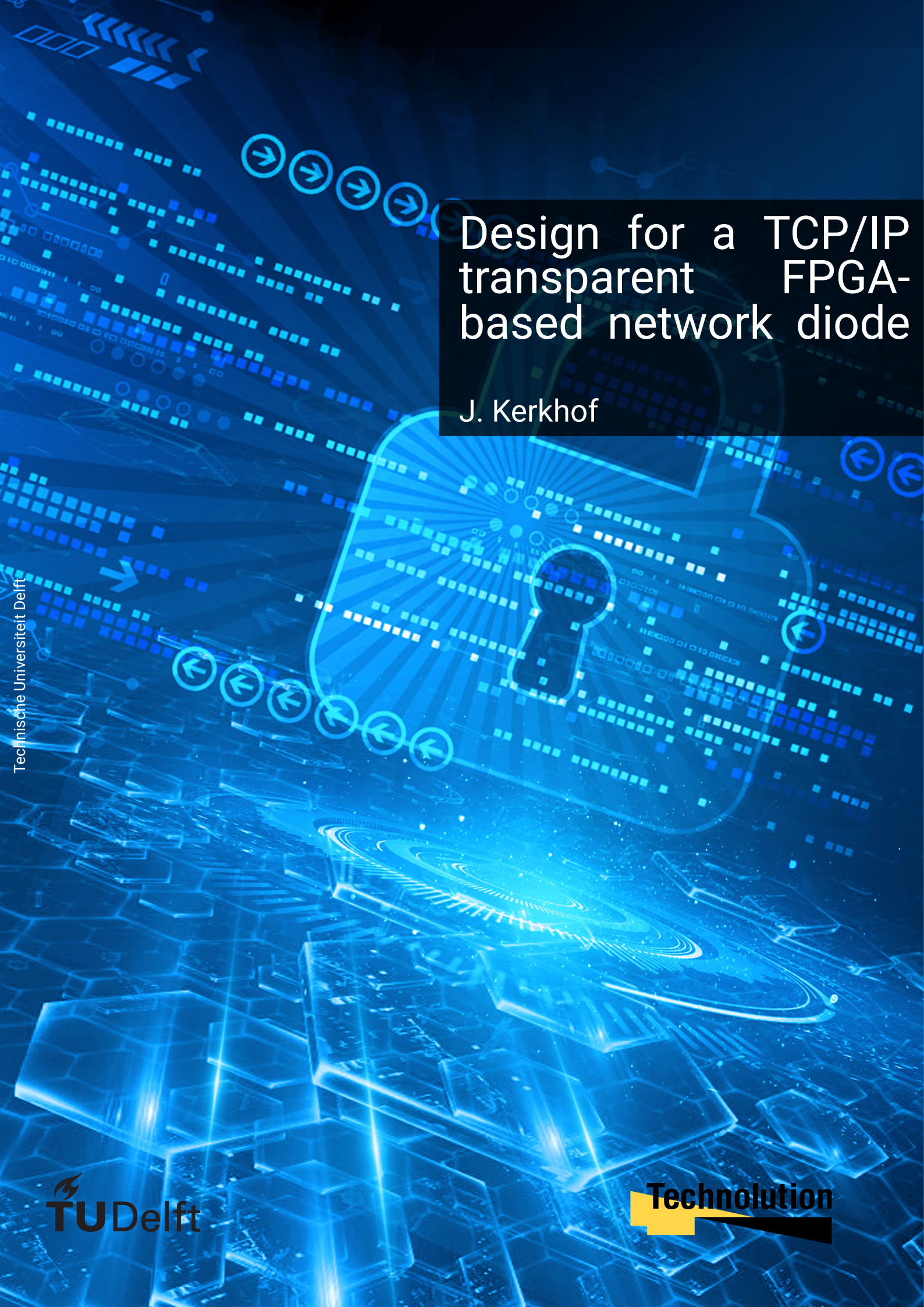


Design for a TCP/IP transparent FPGA-based network diode

J. Kerkhof

Technische Universiteit Delft

TU Delft

Technolution

Design for a TCP/IP transparent FPGA-based network diode

by

J. Kerkhof

to obtain the degree of Master of Science
in Copmuter Engineering
at the Delft University of Technology,

to be defended publicly on January 27, 2020 at 10:00 AM.

Student number:	4232461	
Project duration:	Sept 2, 2019 – Jan 27, 2021	
Thesis committee:	Dr. Ir. J.S.S.M. Wong,	TU Delft, supervisor
	Ir. A.D. Wiersma,	Technolution, daily supervisor
	Dr. Ir. A.J. van Genderen	TU Delft
	Dr. Ir. T.G.R.M. van Leuken	TU Delft

This thesis is confidential and cannot be made public until January 27, 2022.

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Technolution

Acknowledgements

This thesis contains my final project of the Computer Engineering master at the faculty of EEMCS, Delft University of Technology. This marks the end of my time as a student at the TU Delft. During this period, I met new friends and gained valuable experiences along the path.

I would like to thank Technolution, for providing me the opportunity to work on my graduate research project under their guidance. I am grateful for the advice and feedback from my daily supervisor, Ard Wiersma. Even during the lockdown, I was welcome to have personal meetings.

From Delft University of Technology, I would like to show my appreciation to my supervisor, Dr.ir. Stephan Wong, for all the time and effort he put in our meetings. His insights and feedback helped me keep the thesis project progressing and guided me in determining the final direction of this thesis.

Finally, I would like to thank all my friends, roommates and family for their mental support and positive words. It has been a long and rough year, and I would like to thank you for seeing through these tough times. I would like to express my particular appreciation to Colin, for giving his clear-sighted view on my project.

Last, but certainly not least, I would like to show my gratitude to Judy, who helped me through my toughest times and was always able to put a smile on my face. Thank you for all your support this year and I hope we have many, more beautiful, years to come.

J. Kerkhof
Delft, January 2021

Abstract

The urgency for high-security products for industrial networks is increasing as malicious hackers are improving their accessibility tools. A common practice for a company to protect its sensitive data is network segmentation. The network is segmented in different domains with distinctive security levels. The sensitive data is stored and managed within the domain of the highest security level. To access this domain from another domain of a lower security level, a highly reliable connection is required. You want to have full control over the incoming and outgoing data flow between these network segments. A variety of solutions provide a highly-secured connection to link those segments which differ in range of features and control. An upcoming trend is the network diode. This device will allow data flow in only one direction. All the flows going into the opposite direction are being blocked. However, to feature an arbitrary flow between two network segments, the diode should consist of a numerous amount of properties.

To narrow down the optional features a network diode should provide, this thesis will focus on TCP streams. TCP is one of the most common protocols used in internet traffic. Furthermore, TCP is a challenging protocol as it is a connection-oriented bidirectional protocol, which is intrinsic controversial with the concept of the data diode. To ensure the security of the data diode, this thesis will focus on a complete hardware design of the data diode. Software inside the data diode is still a risk for a security breach. This thesis will investigate the critical operations of TCP to implement them in the data diode. The aim is to utilise TCP's characteristic operations of the acknowledgement managing, the sliding window system, the congestion control algorithm, and explores the advantage of existing TCP options.

To evaluate the feasibility of a high performance data diode featuring TCP, the system is broken down to project the behaviour of a TCP stream on a one-way connection device. This results in two separate TCP connections, with only the precious data as common shared information. This model requires a buffer at one side of the diode to transmit data in a TCP stream. To analyse and examine the influence of the diode configuration to the size of the buffer, a diode module is created to simulate in a OMNeT++ environment. From this simulation tool, a minimal set of parameters can be extracted that are essential to configure the data diode. With the assumption of having control on the network management at the trusted side of the diode, a configuration without a congestion control algorithm and without adding radical TCP options is recommended to minimise the required buffer size.

Thereafter, this thesis proposes a high-level hardware design to implement the data diode on a hardware project. The design focuses on the high data rate which should be available to satisfy the data diode's requirements. Finally, this thesis concludes with an elaboration on the assumptions of the limitations of the network environments and recommends features to implement in future work.

Contents

1 Introduction

1.1. Assignment

The dependency on reliable communication infrastructure has never been as strong as today. Cyber criminality is rapidly growing and it is threatening our daily lives in more serious forms every day. Figure 1.1 depicts the number of cyber attacks on government agencies, defense, high tech companies, or economic crimes with losses of more than a million dollars registered by the CSIS[18]. those are concerning numbers and would explain the necessary attention for data security.

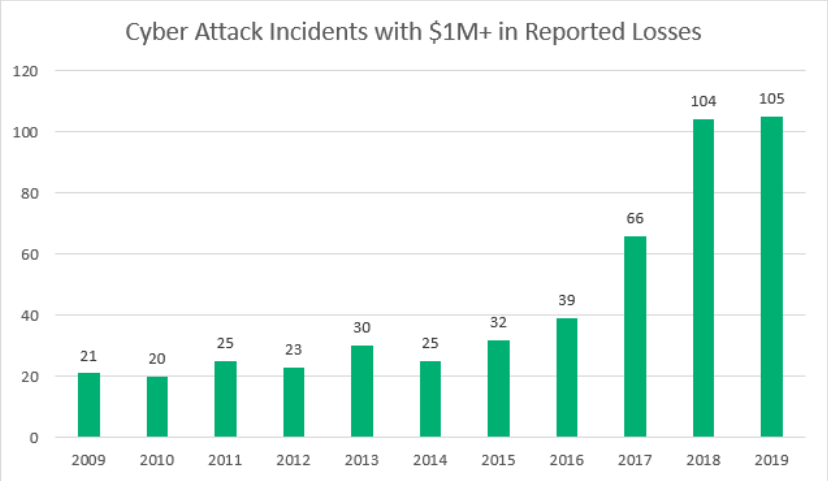


Figure 1.1: Number of cyber attacks over the past 10 years [18]

To protect their sensitive data, companies are segmenting their network into different domains. Each domain consists of systems and networks with the same functionality and the same security level. An example of this is depicted in Figure 1.2. This is the first step to separate your company network from the World Wide Web (www). Each domain is assigned to a different security level. Each security level has different permissions and has a specific user group that can access this level. Most valuable data is located in the domain with the highest security level. All data within each security level is in its own domain and the information can flow freely between all devices known to share the same security level, but it cannot ever leave that domain without verifying that the receiving user/device has permission to access that data. The communication between the domains poses a risk in protecting the data from users and devices that do not have permission. Hackers could get unintentional access or upload malicious software to sensitive databases or critical systems.

To properly control the access to a domain, each domain must be completely separated from each other, in such a way that every in- and output flows through a single point of access. The information flow between those domains has to be handled with extreme care. A typical method is placing a firewall between the domains. Firewalls are in control of the data flowing between the domains. They can be configured to choose

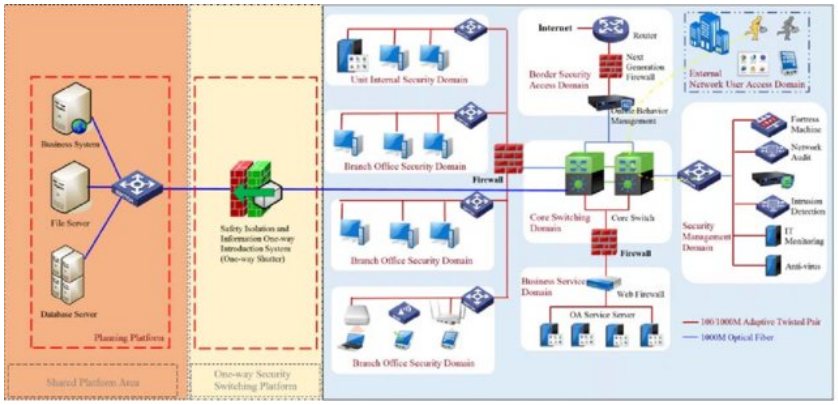


Figure 1.2: Example of network segmentation in different security domains

for each piece of data to go through or being blocked. In this way, firewalls are able to filter traffic based on address, protocol, size, frequency, etc. Firewalls have the power to block very specific data flows. However, firewalls are a common software solution to restrict access, which leaves them vulnerable to cyber attacks as well. Simplified, a firewall is a set of rules which apply to all passing data. If configured correctly, all hazardous traffic flows through the firewall and will be verified. The rules can be configured via a managing access point. Via this method, an intruder can get access to this list of rules and permit access to his malicious content.

A different way to control access is by creating the separation between domains in hardware. Hardware is not vulnerable to cyber attacks, but has difficulties processing more complex functions, like filtering on network perimeters. A potential solution, however, is creating a one-way connection, established in hardware, thereby eliminating the outflow of data from one domain entirely. Such a product would be called a data diode. At Technolution, they are developing high-end security products for domain separation, including data diodes and cross-domain solutions. By placing a **data diode** between two domains, a one-way connection is forced and makes it physically impossible to transmit information in both directions. A data diode provides a highly secured connection by blocking all traffic from one side of the connection. This diode separates the two domains physically in one direction. An application example is depicted in Figure 1.3. The domain on the left side represents the drilling side of an oil refinery. This is where the machines are running and controlled by a workstation. You want to access information about the machines remotely at the head office to check things like maintenance state, production value, etc. Only the network of the head office is connected to the World Wide Web and thus accessible by anyone. Therefore, the machines should not be able to be controlled remotely from the head office, because eventually hackers could access the head office's network. By placing a data diode between the drilling site and the head office, the drilling site can still send their measurements to the head office, but the head office is unable to adjust any settings at the drilling site.

Internet traffic has a lot of mechanisms to transmit information from device A to device B. Most of those mechanisms are standardised by protocols to agree on how data is sent. Currently, over 85% of all data worldwide is transferred by TCP [42]. Nowadays, many applications cannot operate without using TCP. Therefore, the data diode must comply with TCP. However, TCP is a connection-oriented problem and will have issues with a data diode in between the connection link. TCP requires a bidirectional flow to operate, which is intrinsically not possible once a data diode is inserted.

A frequently used solution is to perform this conversion in proxy PC's. At the incoming side of the diode, the TCP stream is broken down to a unidirectional information stream that can be sent over the diode. At the other side of the diode, another proxy PC will convert this back to a new TCP stream. However, this solution requires a lot of maintenance on the proxy PC's and is costly. Both proxies are installed at different locations and need configuration updates. Moreover, during the conversion of the TCP flow, irredeemable data loss occurs due to insufficient buffering or other software errors in the proxy PC.

Instead of having two proxy PC's and a data diode, a single-board design is desired. This should improve maintenance costs and time. A hardware solution is required to solve the data loss errors during buffering.

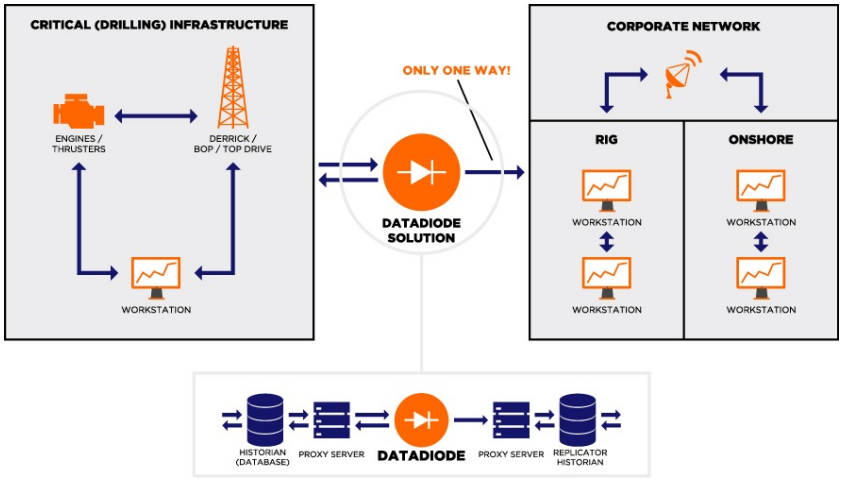


Figure 1.3: Example application of data diode

FPGA technology allows to implement the complex functionality of the TCP stream conversion as well as high-speed data handling in a secure environment. Embedded softcore CPUs can be combined with dedicated hardware processes to achieve this.

1.2. Problem Statement

From the previous section, the following research question was derived:

To what extent is it possible to implement a network diode on an FPGA under realistic network environments, using the Transmission Control Protocol?

In order to work towards a solution, this main research question can be broken down into smaller parts. The implementation of the network diode focuses on three main aspects:

1. The design should be convenient to implement on an FPGA
2. The network diode must support TCP
3. The implementation of the network diode should be verified to operate under realistic network environments

First of all, the possibility to implement the diode on an FPGA should be explored. The main focus is to minimise resources because those are expensive. There will be an examination of the resources and research should be done to what extent those could be minimised. Second, to ensure the network diode supports the use of TCP, it is of great importance to investigate what modifications should be done to convert the bidirectional data flow to a one-directional one. Consequently, TCP supports features to improve the Quality of Service, such as flow control and congestion avoidance. There will be examined which of those features can be implemented, and how they can be implemented. Third, the diode must operate under realistic network environments while still achieving high performance. Therefore, it is vital to define what high performance of the diode is, how this could be analysed, and what the bottlenecks are in terms of high performance. Fourth and last, the diode must be able to maintain multiple connections. The diode should be able to select channels for the specific connections and identify the addresses on both sides.

1.3. Project goals

The goal of this thesis is to present a high-level hardware design for quick implementation of a network diode on an FPGA, supporting the network features of TCP, providing high performance and minimising resource utilisation. To achieve this goal, sequential subgoals are defined.

Goal 1: Define design requirements To work towards a solution, the requirements for a design determine the focus during development. This will assist in selecting the significant parameters of the system. In the

previous section, four main aspects are described. The first is to minimise resource utilisation. Second, TCP should be supported, so the goal is to maintain the reliability of the connection. Third, the diode should achieve high data flow, and fourth, it should feature multiple channels, both as requested by Technolution. Summarising, the design requirements are:

1. **Requirement 1:** Minimise resource utilisation
2. **Requirement 2:** Maintain a reliable connection
3. **Requirement 3:** Maximise data flow to 10 Gbps
4. **Requirement 4:** Manage multiple connections simultaneously

Goal 2: Develop a model for system evaluation
In the previous section, it is stated the question of how to investigate the high performance of the diode. What the definition of high performance is depends on the requirements, which are stated in the previous goal. No standard method exists to measure and analyse the performance. Our goal is to create a suitable testing and developing environment for the network diode. This platform should support settings on the diode configuration, changing TCP settings, and simulate realistic network traffic.

Goal 3: Design an accurate block diagram
Finally, this goal entails creating a high-level hardware design for quick implementation of the network diode. Knowing the implementation will be done on an FPGA, this design should be simple to convert to a VHDL or RTL project. A visual block diagram, assisted by an accurate functional description should outline a modular design for effortless implementation.

1.4. Methodology

The methodology to formulate an answer to the problem statement of this thesis are described as a step-by-step plan.

1. **Research for background information**
   Look at related work on high-end security products for domain separations. Acquire information from implementations which are already on the market. Find proposed methods for network diode implementations. For those solutions, check for their tasks to the distribution in software and hardware. Research diodes implementing TCP and find what features are supported.

   Further, research the theoretical background. Examine the basic operations of TCP and research its typical features and characteristics are. Research what transmission units it uses, what part of the bidirectional flow is essential and what opportunities TCP provides.

2. **Create an analysis model**
   To develop the desired configuration for the implementation, a proper model is essential. With the knowledge gained from the previous step, a simplified model can be formulated from a small set of formulas. From the design requirements stated in the second goal of Section 1.3, it is possible to define the important parameters of this model.

3. **Research for parameter analysis**
   Look for research on those set of parameters and check if methods are determined to calculate and predict on those values. Verify if those methods are applicable to this system or else if they could be modified if necessary.

4. **Simulation software selection**
   The results of aforementioned steps must be verified whether they are applicable in a real scenario. Therefore, the parameters from previous steps must be tested in a simulation. Simulation software is imperative to analyse those parameters. A suitable simulation software should be selected which is capable to test all possible features of the diode and the network traffic.

5. **Configure a simulation module**
   Create a module in the simulation tool which is able to configure all possible features and settings of the diode as well as the network environment. Set up a test plan for finding the optimal set of parameters.

6. **Evaluate results**
   Gather all results from tests executed at the previous step. Analyse the results and reflect them to the design requirements as well as the goals of this project. If analysis demands extra tests to be performed, step 5 and 6 could be repeated.

7. **Develop a high-level hardware design**
   Based on the results of the previous step, a high-level hardware design could be developed. Both sides of the network are separate and thus require different configurations.

1.5. Overview of the thesis

This thesis is structured as follows:

- **Chapter 2**
  This chapter discusses the theoretical background of this thesis. In Section 2.1, the current developments in the industry regarding domain separation products, and especially data diodes, will be discussed . The most promising products will be highlighted and their features will be discussed. Subsequently, in Section 2.2, the theoretical background of the design of our thesis will be explained. In particular, all features and characteristics of TCP will be discussed with a particular focus on their feasibility for implementation.

- **Chapter 3**
  Chapter 3 will explain the first approach to analyse the system. In Section 3.1, he behaviour of the system will be analysed using a simplified model. From that model, the most important parameters are selected for further analysis. To study the effect of those parameters on the behaviour of the system, a suitable simulation tool is required. In Section 3.2, an extensive comparison between simulation software is presented.

- **Chapter 4**
  After a selection of the most suitable software has been made, this chapter introduces new modules representing the network diode. This models are verified and implemented in the selected software package. From those modules, the system model as proposed in Chapter 3 will be built. For this simulation model, a sequential order for testing the set of parameters is proposed. This testing order is based on priority and dependency.

- **Chapter 5**
  The results from the tests as described in the previous chapter will be analysed in this chapter. The effects of the parameters on the performance will be discussed. Furthermore, the results will be reflected on the goals and design requirements as stated in Section 1.3. The results will be composed to a recommended set of parameters, selected for each side of the diode.

- **Chapter 6**
  A high-level hardware design will be presented in Chapter 6, together with the parameter set gained from the previous chapter and the functionality of TCP itself. The design will be explained in the form of a block diagram. The results from previous chapters are used as the backbone of the configuration of the discussed design.

- **Chapter 7**
  Finally, the last chapter contains the conclusion of this thesis. All contributions will be reflected on the achievements of this project and a discussion about the future work is proposed.

2
Background

Before starting to develop a model or defining any important parameters, first the background of the project must be researched. An overview of similar products available on the market will give some insight into the bottlenecks of this project. Of course the operating mechanisms of the system as well as from the protocol will be explored. Knowing these mechanisms will give a clear perspective how to develop the simulation model and what features of TCP are worth researching in more detail.

In this chapter, at first the related work will be discussed in Section 2.1. We will look at current developments in the industry. From these solutions we are going to identify if they do solve the problems as stated in Section 1.2. With this research we want to determine what features are missing or can be improved in my solution. In Section 2.2 I am going to explain the transmission control protocol. The most important features will be described. As TCP is a complex protocol with many operating mechanisms and available extensions, I will highlight the feasible options regarding the capabilities of the network diode.

2.1. Related Work

As described in Chapter 1, the dependency on reliable communication infrastructure has never been as strong as today. Many companies are investing in high-end security products to protect their networks from intruders. The data diode is introduced to separate two domains physically in one direction.

The first data diodes were developed by governmental and national defence organisations. Because these organisations work with classified information, they want to protect their network with the highest security. Over the years, standardisation of this technology has occurred. Since 2010, regulators are encouraging the use of unidirectional products in sectors working with critical information throughout the world. Industries like oil refineries, railway systems, nuclear reactors, and energy suppliers should protect their vital systems by such technology [11, 28, 63].

Because the data diode blocks one direction of the information flow, it has two use cases. The first is when the flow is allowed from a high secure domain to a less secure domain. For example, when a power plant needs to transfer measurement data from the power plant site to the head office. A second use case is when the information flow is allowed from a less secure domain to a high security domain. An example of this is when classified data has to be stored in a location which is not connected to the internet. Traditional way was by separating this data server by an "air-gap". The data has then to be transferred by human interaction with a USB-stick for example. A data diode fixes this problem and the data can be securely transferred to the database. There are also a few cases where data flow in both ways is controlled. This is normally done by placing two data diodes at the edge of the security domains.

Today, 4 different types are defined [14]:

1. **One-Way cable assembly**
   Separating two network domains can be as simple as cutting the receive/transmit wire between the two devices. This will only physically allow one computer that can transmit and one that the other can only receive. This is still not entirely safe as the connections are not galvanically separated.

2. **Firewall-Enabled Policy**
   A firewall could be considered a data diode if it is configured in such a way. Firewalls are pieces software running at the edge of a network controlling the data flow. They can be configured to only pass through data in one direction only and block all incoming traffic.

3. **Unidirectional Gateway**
   The unidirectional gateway is implemented by placing specific pieces of hardware at the sender and the receiver side with a single one-way connection between them. At each side, a proxy PC is placed before the gateway hardware. The proxies serve as converter from two-way protocols to one-way data transfer. They assist in identifying packet information, negotiating one-way transfers, transmitting packets, receiving packets, and rebuilding the packets. Besides this, they take care of the expected confirmations and handshakes of the original network.

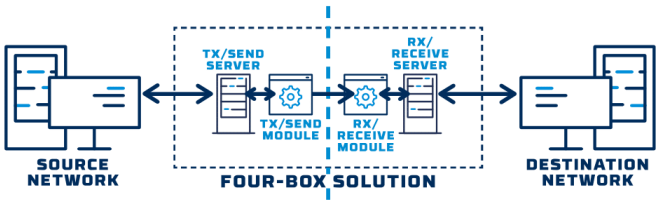


Figure 2.1: Graphical representation of the four-box solution of a data diode [14]

This solution does need four devices, two on both sides. The proxies configurations needs to be aligned. Agreements on types of communication, protocols and data flow has to be configured at separate locations. For that reason it is difficult to make adjustments to the functionality of this system.

4. **Intelligent Data Diode**
   The most advanced data diode implements all functionality inside one single device. The one-way connection is realised by an electrical separated connection. It is completely impossible for the receiving side to transfer information while the sending side is incapable of receiving any information.

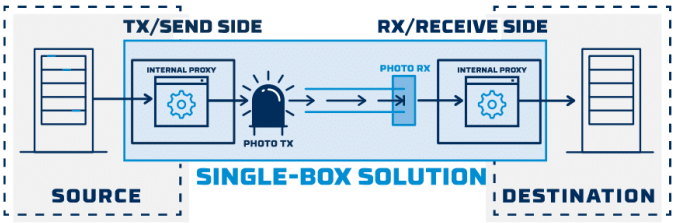


Figure 2.2: Graphical representation of the single box solution of a data diode [14]

This solution is provided together with coupled proxy servers built-in the same device. The proxy servers make it possible for supporting two-way protocols as well as multiple data flows and data types simultaneously flowing through one device.

2.1.1. State-of-the-art data diodes

Securing data flows between network segments by placing one-way connections is becoming the dominant solution. Providing a hardware solution has numerous advances comparing it to software solutions with regards to security. However, software costs less effort to adapt to innovations in the network sector and is more flexible in contrast to multiple protocols, data flow, and types. Many data diode implementations are available on the market today. Both unidirectional gateways and intelligent data diodes are being developed. We will highlight the most promising designs.

Unidirectional gateways

FOX-IT
Fox-IT's crypto datadiode has been awarded with the highest grade of security products by the AIVD and the NBV as well as received the EAL7+ certificate [26]. However, there are no specifications about the implementation and the features of this diode.

Rolloos
Rolloos implements the unidirectional gateway for large data transfers. A hardware device manages the one-way connection using the technology of light emitters. Two proxy servers are placed at both sides to convert the bidirectional flow to a unidirectional flow [56]. Copies of the databases are stored at both sides of the diode where information is transferred from the construction site to the corporation's office. This design can be directly installed in any industry sector network.

Arbit
The Arbit Data Diode moves data from an inscure network to a secure network ensuring that no data is able to flow back [7]. The one-way connection is established by a single fiber-optic cable. Two dedicated servers are attached to this cable, which are responsible for the transmission. This diode supports multiple data channels and most convenient protocols, including TCP. The light-emitting diode has a maximum transmission speed of 690Mbps but is limited in maximum file size.

BAE
The BAE Systems Data Diode Solution has an EAL7+ certificate and is National Cross Domain Strategy Management baseline approved in the UK [31]. They promote a Raise-the-Bar compliant, one-way transfer which is established by an optical fiber. The transmission is supported by utilising software predicated on Red Hat Enterprise Linux software. It features multiple connections and protocols, including TCP. The data diode datasheet promises a throughput up to 100 Gbps.

Siemens
The Siemens Data Capture Unit is a one-of-a-kind data diode solution [67]. The Data Capture Unit (DCU) is able to monitor all passing traffic. Independent of the protocol, it can securely capture all bit streams and set up a data flow from a critical network to the open network. The DCU is implemented in hardware and uses technology used in the defence industry for many years. The DCU is assisted by two one-way gateways implemented in software. They proclaim to be the most cost-effective and user-friendly solution for secure cross-domain products today.

VADO
The VADO's One Way Data Diode consists of two units, one containing a photo-diode emitter and the other containing a photo-diode receiver. The distinction between this diode with the other solutions is the implementation of the proxy functionality. The physical device is a stand-alone machine. VADO Agents can be installed on OS Virtual Machine, and no additional NIC needs to be attached to the OS[66]. Their solution features speed up to 1Gbps and is focused on file transfer protocols, not including TCP. The solution is flexible and should fit in every client's network topology.

Waterfall
The Waterfall Unidirectional Security Gateways should replace firewalls in industrial network environments. It enables safe IT/OT integration together with real-time monitoring from the industrial network. The Unidirectional Gateway provides hardware-enforced network perimeter protection [60]. The one-way connection is realised by two photo diodes and an optical fiber cable. The hardware is supported by software components consisting of industrial application software connectors by Waterfall. These software connectors feature applications like historian databases, remote monitoring, file transfers, etc. Waterfall's technology provides a plug-and-play replacement for firewalls.

Filbico
The ZNO Data Diode is designed to protect critical information classified under NATO SECRET [19]. The design consists of four components. The transmitting data flow controller, transmitting separator, receiving

separator, and the receiving data flow controller. The transmitting and receiving separators are connected by an one-way fibre-optic interface(SPS-1G). The data flow controllers provide hardware and software interfaces and have integrated software. This software is dedicated for filtering and transferring permitted information through the SPS-interface. The software runs on a Red HAt Enterprise Linux 7 system. The system supports many transfer protocols, including TCP, and reach data rates of 1 Gbps.

DeepSecure
The Data Diode of DeepSecure is realised by a pair of servers and a uni-directional fibre optic link. The system is very user-friendly with an intuitive GUI and simple setup. The diode supports multiple channels and protocols, including (framed) TCP [59]. The device has a 1 Gb NIC interface, but the actual throughput performance is not stated.

PrimeDiode 3010
The PrimeDiode of Technolution provides a high-security network device [64]. The diode is certified SECRET by the dutch cyber-security authorities. The device is a uni-directional gateway, that allows data flow in only one direction. There is no electrical connection between the in- and output sockets of the device. To ensure the network separation, only the RX-connections are available at the receiver side, while only the TX-connections are available at the transmission side. Both networks connected to the Primediode require a proxy to transfer data. The proxies can be configured to the customer needs.

Intelligent data diodes

Owl cyber security
Owl cyber security implementation of the intelligent data diode includes an electrical isolated one-way communication with the use of photo-diodes [14]. This single-box solution is provided with two additional proxy servers at both sides of the diode. Each proxy is only accessible by the side it is located and both proxies are configured and managed separately.

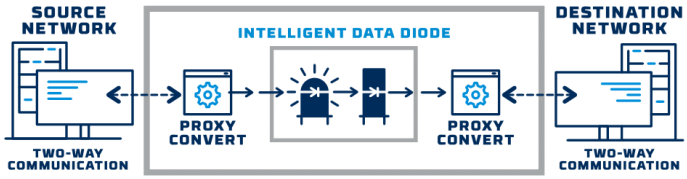


Figure 2.3: Intelligent data diode by Owl Cyber Securtiy [14]

All functionality of the diode is designed to operate in a one-way fashion. The proxies around the photo-diodes are creating packets to use the ATM (Asynchronous Transfer Mode) protocol [25]. ATM is designed by telecommunication companies for digital transmission of multiple types of traffic using a one-way connection. This protocol is designed for sending data types of video and voice messages, which require low latency, without retransmissions or bidirectional communication. The proxy servers convert packets from their original protocol (TCP, UDP, etc.) to ATM packets. This method has the additional benefit that only payload is transmitted over the one-way connection, not including any addressable information. This solution features rates at up to 10 Gbps with a packet transfer latency of 2 milliseconds or less. The disadvantage of this design is the fact that the proxies have to be configured and maintained.

InfoDas
The SDoT Diode is the fastest software-based Data Diode in the world with global classified SECRET accreditation proclaimed by InfoDas[30]. They do not use a fibre optic cable to enforce the physical separation of the security domains. The SDoT diode ensures a logical separation of networks without a return channel due to its unique security architecture. No details of this architecture are mentioned. The diode allows fast and high-performance unidirectional data transfers via numerous protocols in a compact form factor. This diode features data rates up to 9.1 Gbps and multiple protocols, including TCP.

Wizlan
The VIT-400 by Wizlan features a network-to-network fibre-optic coupled diode and approved by the Israeli National Information Security Authorisation (NISA)[70]. The device provides a complete isolation of the networks, including the fibre-optic connection as well as separated power supplies. The VIT-400 is a complete hardware solution. VectorIT software applications need to run on the connected endpoints of the diode. This software is dedicated to transfer data over the unidirectional link. VectorIT features link verification of the incoming endpoint. The VIT-400 supports transfer protocols, including TCP and has 100Base-T ports for the interfaces.

Rovenma
Rovenma's Kindi Data Diode creates a fully closed unidirectional data transfer protocol that cannot be routed between networks, enabling complete isolation of the two networks [58]. A receiver and sender device are placed at the source and destination network. The devices are connected by a one-way SFP connection and provide point-to-point or point-to-multipoint delivery. At the endpoints, a specialised Data Pump Application is offered. The device features a maximum 2μsec latency and a throughput of 10 Gbps. Extra security can be implemented by integrated hardware encryption. The supported protocols are not stated.

Advenica
The SecuriCDS Data Diode allows for real-time, unidirectional information exchange between networks. The diode provides the separation of the two networks by using optical transmitters and receivers, and even separated power supplies [1]. The data diode is supplied with integrated proxy servers which will handle the transfer via common communication protocols, including TCP. The SecuriCDS DD1000i is approved by multiple national organisations to handle information classified TOP SECRET. The diode provides the opportunity to customise the proxy services to your own needs. The software development kit lets two separate administrators customise their side of the network diode. The diode is supplied with interfaces handling up to 1 Gbps data transfers.

dataflowx
DataFlowX deploys a cross-domain security gateway solution, isolating networks from the physical layer. The diode is not only an protocol-based or file-type-based one-way connection, but also features data analysis and modifications before sending[20]. It is even possible to integrate third-party security solutions like intrusion preventing systems, sandboxes or threat intelligence feeds. The network administrator can do any control, modification and verification over objects passing the diode.

Controlled Interfaces
Controlled Interfaces provide optical products, which can be installed as plug-and-play. They claim that it can be attached to every device in your network and therefore supporting data rates up to 100G or even more[17]. The device is adjustable to different optical hardware, software and applications. Therefore, it supports many protocols (including TCP), many hardware architectures, and many device configurations. No media conversion is applied, because their product is all-optical. You still need controlling devices to transfer data.

Oakdoor
Oakdoor provides an all-in-one cyber security unit, including two data diodes and integrated processing hardware to provide highly controlled unidirectional data flows [16]. The integrated hardware runs software for safe browsing and safe data established by the Oakdoor Gateway platform. The diode does feature throughput of 1 Gbps and does support protocol conversion, but they do not specify which protocols.

2.1.2. Conclusion
As the importance of digital information grows and even more systems are getting interconnected. Companies have critical networks and want to protect these with high priority. Data diodes are replacing the "air-gaps" to reduce the large latency. Firewalls are being replaced by data diodes to provide more security. Two main data diode implementations are on the market. The unidirectional gateway, which provides a secure one-way connection. Only this device needs two additional servers installed at the edges of the network domains. The data diode transfers the data in between these devices. This provides the ability to let the consumer configure the proxies at both sides and usually comes with an excellent user interface. This solution

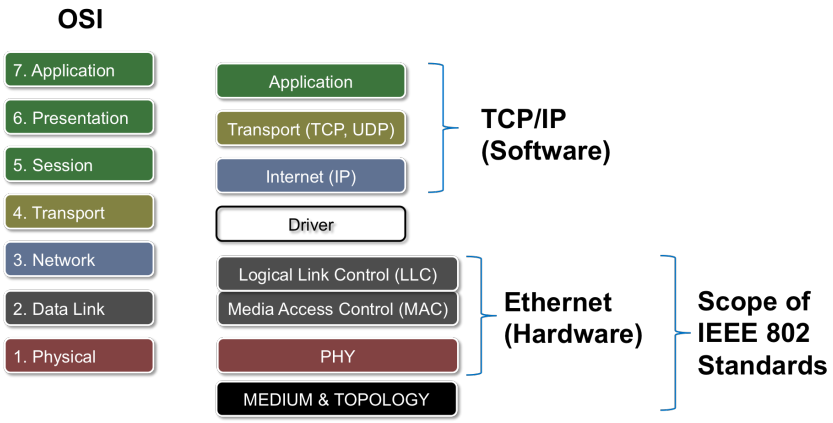


Figure 2.4: Common internet implementation of the OSI model for standard hosts [49]

often has a high throughput and is configurable to what protocols it supports. However, the proxy servers need maintenance and installation is expensive. The other solution is denoted as the intelligent data diode, because it is only one device to install. The one-way connection is preferably established by two photo diodes and needs a specific controller for this. The conversion from a two-way connection to the one-way connection is handled by on-board software utilisation. It depends on the manufacturer if this is reconfigurable by the end-user or pre-configured at installation. The advertisements of products are very promising with high-speed data transfers. Not all presented solutions are supportive for using the TCP protocol. Only most companies do not reveal many in-depth technical aspects. They promote with hardware-based solutions, yet no solution was developed using only hardware in their implementation.

2.2. Theoretical Background of the TCP/IP suite

Before starting to design the TCP/IP diode, the functionality should be explored in all its aspects. The main function is the assurance of a one-way connection. The more complex functionality is introduced by the requirement for supporting network protocols. Network protocols provide rules, mechanisms, algorithms, and specified frames to improve the quality of service of a connection. All standardised protocols are constructed of multiple layers as described by the OSI-model.

The OSI model is a conceptual model used for all electronic telecommunications [21, 50]. This model defines 7 specific layers to standardise communication protocols. Every layer is an abstraction of the functionality of transmission. A graphical representation of all layers is presented in Figure 2.4. The physical layer consists of hardware devices which are able to send and receive digital data flow. In our case, the physical layer will be realised by the network component available on the FPGA. The data link layer defines methods to connect these devices and identify them. This layer is Ethernet. Many devices form a network, and to find a route between all the device, the network layer consists of network protocols, which is IPv4 in our case. To improve the reliability and define segments in chunks of data, protocols in the transport layer are defined. The upper three layers define methods for interpreting and processing the data, which are not important for the diode. To send information from host to host, applications will pack data and send them to the lower layers. Each layer will add some information of the data, connection, communication, or information for the other layers. A common partition of layers handled in hardware and layers handled in software is depicted in Figure 2.4. In most standard network devices the physical and data link layers are implemented in hardware (e.g., Network Interface Cards). The upper layers will be handled in software together with a suitable driver for the lower layer. The focus of this thesis is on the Transmission Control Protocol.

2.2.1. Transmission Control Protocol

The key feature of the transmission control protocol is its ability to maintain a reliable bi-directional communication. This protocol is most common in current internet applications [2, 42, 43]. A key factor for the robustness of this protocol is that TCP acknowledges all transmitted data, so the sender will know the data arrived. TCP is a connection-oriented protocol, which means that before any data is sent between two hosts, a connection has to be set up. TCP uses segments as transmission units. Because of the wide variety of characteristics of this protocol, we partitioned the operations of this protocol based on functionality. The protocol

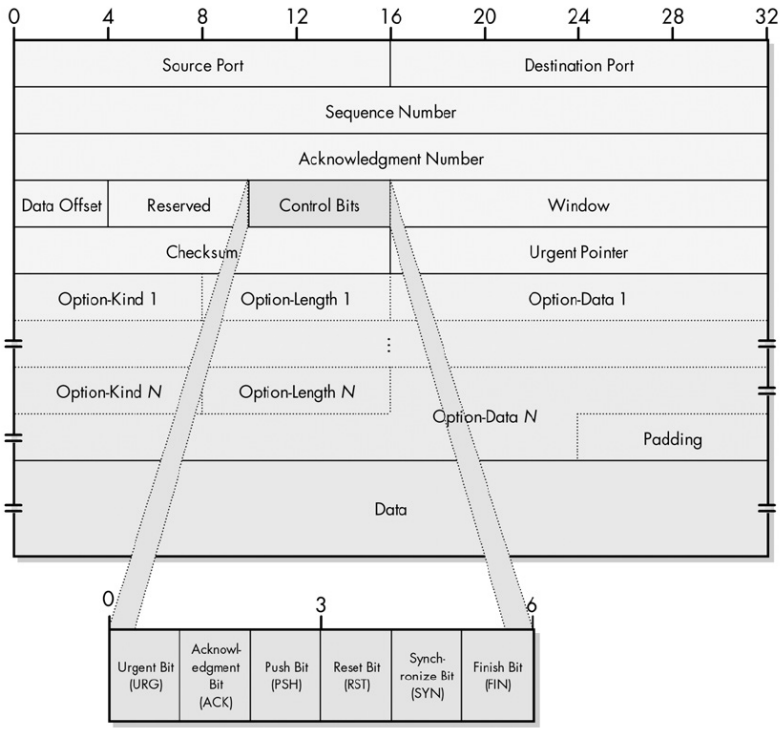


Figure 2.5: Overview of TCP segment [40](fig. 48-1)

has a defined header which will be attached to the packets, further referred to as segments. These segment contains information fields to improve the quality of the data flow. TCP is a connection-oriented protocol, so it provides mechanisms to establish a reliable connection. The reliability of the protocol is supported by mechanisms and algorithms to verify segments have arrived and what to do if packets get lost. To send packets at a rate that suits the receiver, TCP is able to manage the flow control in both directions. Along with this, TCP takes into account other traffic using the same resources and provides methods and algorithms to prevent congestion in the network.

2.2.2. TCP Header

Every TCP segment contains a header with information regarding the data or the connection. A overview of this header can be seen in Figure 2.5. Each time a segment arrives, first all header fields will be processed before processing the segment. The specifications for these header fields are described below.

1. **Source/Destination Port**
   The first part of the TCP header contains the source and destinations ports. These ports define for which application the data is. An example of this port is port 22, which is used for SSH.

2. **Sequence / Acknowledgement number**
   The sequence number is used for identification of the TCP segments and is one of the key elements for reliability of this protocol. Every segment has its own unique sequence number and are sequential to the data stream. The sequence number is set by the sending host and the receiver will send an acknowledgement with a corresponding acknowledgement number to let the sending host know the segment arrived correctly. The acknowledgement process will be clarified in Section 2.2.5. The acknowledgement number is only valid when the ACK-bit is set high in the control flags.

3. **Data offset**
   This part of the segment tells how long the header of the TCP segment is in number of 32-bit words. Therefor it also indicates by how many words the start of the data is offset from the start of the TCP segment.

4. **Control flags**
   These flags or bits are determining the control flow of the connection. These *control flags* indicate the kind of segment which is received. Currently there are 8 bits commonly used [43].

   - ECE
     The IP layer is able to notify for congestion events. The ECN (Explicit Congestion Notification) Echo flag is used to negotiate the use of ECN during synchronisation phase. Also the flag will be used as indication of a Congestion Event while connection is established.
   - CWR
     The Congestion Window Reduced flag will indicate a reduction of the congestion window in response to a ECN flagged segment.
   - URG
     The Urgent flag indicates the Urgent field in the segment header is valid. The urgent field is a pointer to the segment which needs priority.
   - ACK
     The Acknowledgement flag indicates that the acknowledgement number in the segment is valid. This flag should always be set except for the first segment sent in a connection. Thereafter, each segment should contain a valid acknowledgement number for an established connection.
   - PSH
     The Push flag indicates indicates the segment needs priority. Instead of buffering data before sending them to the upper layer (e.g., an application), this flag indicates the segment should be pushed as soon as possible.
   - RST
     A Reset flag indicates an error occurred and notifies the receiver to reset the connection. The reset flag is set when an inappropriate segment is received or to terminate a connection abruptly.
   - SYN
     The Synchronisation flag is only set in the first segment. This indicates the start of the *3-way handshake* and starts the negotiation. This negotiation consists of exchanging sequence and acknowledgement numbers as well as an agreement to what options are being used and not. The *3-way handshake* will be discussed in Section 2.2.3.
   - FIN
     The Finish flag indicates the receiver that the sender has no more date to send and is ready to terminate the connection. The receiver will acknowledge this and will also send a Finish flag when it has received all data and has nothing left to send.

5. **Window Size**
   This represents the number of bytes the sender of this segment is capable to receive. This is commonly corresponding to the allocated buffer size of this connection. This is an essential parameter for the *sliding window*, which will be explained in Section 2.2.4

6. **Checksum**
   This is a 16 bit checksum for error detection. This is computed over the entire TCP segment (e.g. header and data), plus a specific pseudo-header. This checksum adds reliability in regards to the transmission as well as the delivery, which will be explained in Section 2.2.5.

7. **Urgent Pointer**
   The urgent pointer is a used for controlling priority data transfer. This pointer refers to the last byte of the urgent data.

8. **Options**
   At the end of the TCP header there is room for options. Currently there are a lot of different TCP options available. These options adds the opportunity to modify or extend the standard protocol. Each option field has the same values. The first byte represents the kind of option. The second byte indicates the length of the option (including the first two bytes). The other bytes are defined by the option itself. The most common options are listed below sorted on kind.

- End of Option list
  This option indicates the end of the option list and consists of one single byte with value 0. This option is only used when the set of options ends before the end of a 32-bit word is reached.
- No Operation
  This option is used to separate options if necessary and consists of one single byte with value 1. The receiver knows there is at least one other option to process when it reads this option.
- Maximum Segment Size
  This option consists of 4 bytes. First with value 4, second indicates the length is 4 bytes. The other two bytes determine the maximum segment size to be sent. This option can only be used in the negotiation phase. Default value for MSS is 536 bytes. The maximum transfer unit for IP is 576 and subtracting 20 bytes for both IP and TCP header will leave 536 for the segment.
- Window Scale
  This option has kind number 3 and has a length of 3 bytes. This option allows devices to use much larger windows than can be specified in the traditional TCP header window field. The value of this options specifies a power of 2 to be multiplied with the value in the window field.
- Timestamp
  This option has kind number 8 and has a length of 10 bytes. This option is designed for high-speed communications. The sequence number field is limited in its size and with high speed communications there is a possibility to cycle through all the available numbers quickly. A timestamp is added with this option to verify no old segment are being acknowledged. This timestamp has another advantage as it helps improving the round trip time estimation.
- Selective Acknowledgement
  This option is first negotiated by option kind number 4 and length 2. Which is the SACK permit option. If both hosts are able to use this option, the selective acknowledgement will improve reliability and flow control of the connection. Standard acknowledgements are cumulative. With SACK the hosts are able to distinguish separate lost blocks of segment by communicating this in the SACK option. This will be discussed in Section 2.2.5 and Appendix A.

2.2.3. Oriented connection

The key feature of TCP is its bi-directional controlled transmission flow. TCP requires two hosts which want to communicate to establish a reliable connection first. This establishment is done by its typical *3-way handshake* [41]. This handshake is executed by the following steps:

1. When Host A wants to set up a connection to Host B it sends a TCP segment with the SYN-flag. This is the only TCP segment without an ACK-flag, because it is the first of all messages. This message is a request from *host(A)*| *port(A)* to open a session to *host(B)*| *port(B)*[43].

2. When this messages arrives at Host B it will sent an ACK to let Host A know the SYN is received correctly. The connection is half-complete. Host B will also send an SYN to Host A for configuration negotiation. This negotiation works like this: host A sends their values for parameters like the Maximum Segment Size and Window Size, whereas host B responds with his own values for these parameters. The lowest value will be chosen so both hosts can operate under these circumstances Also in this negotiation will be agreed upon using TCP options like SACK. Both hosts sends their available options and the options which match will be used. Usually the ACK and the SYN from host B to host A are sent in the same segment.

3. When Host A receives the SYN from Host B it will sent a ACK to Host B and this is the third segment of the 3-way handshake. Now a connection is established and data transfer can start.

Each "way" of the 3-way handshake represents a message sent from host A to B or vice versa. To keep track of what control messages the device is expecting to receive (or has to send), the device maintains in a particular state. Each message from the 3-way handshake will transit the device to a new state. Not only for establishing the connection, but also for closing the connection, different states are defined. The best way

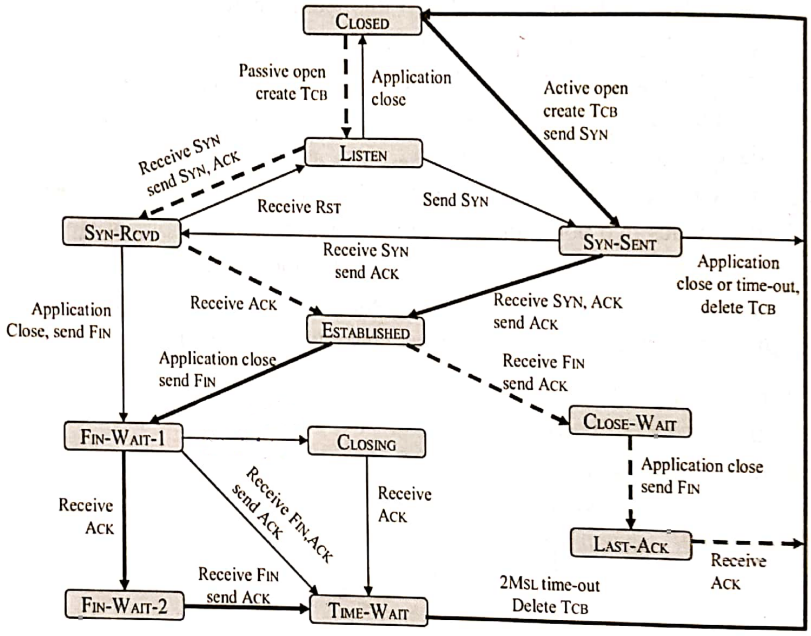


Figure 2.6: TCP State Diagram as stated in RFC 793. Bold and dashed lines shows the normal state trajectory for a client and server respectively. [50]

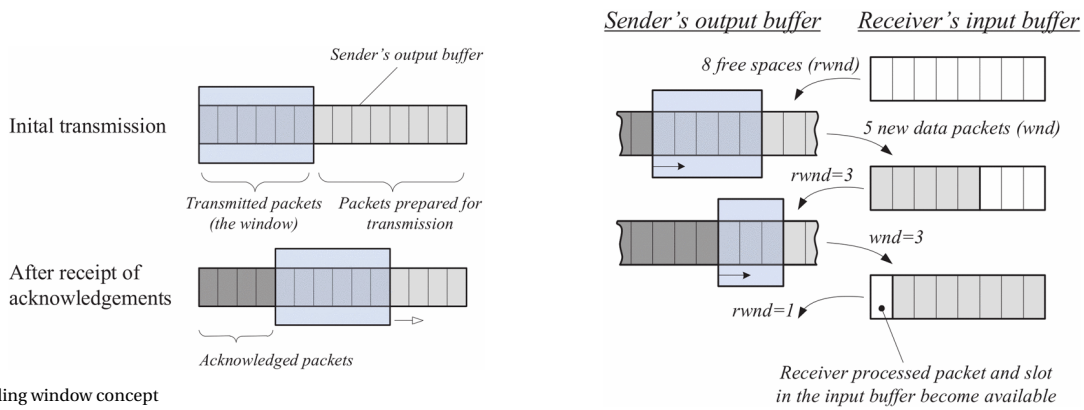
to manage this states and transitions is by implementing a *finite state machine (FSM)*. Figure 2.6 presents an overview of all possible states and which control flags are triggers for changing state. The upper half of Figure 2.6 represents the 3-way handshake. The distinction between a passive open or an active open depends on whether the TCP socket acts as a server or a client. The server will be commonly do a passive open to start in the *Listen* state and then waits for the client to send a request. The client will initiate a connection by sending a SYN first.

Each TCP connection is identified by the combination of the IP addresses of the source and destination together with their port numbers. This combination is unique for every connection and this method allows multiple TCP connections on the same host by multiplexing of the port numbers. These port numbers are assigned to specific applications. HTTP, for example, has port number 80 and SSH has port number 22 [41, 43].

The state and sockets of the connection are being tracked by the *transmission control block* (TCB). The TCB contains information about source and destination addresses, status of the connection, buffers for sending and receiving data, and additional information about the packets being exchanged. For each segment coming by, the values stored in the TCB are being checked and will be updated with new values if necessary.

2.2.4. Flow Control

Once the connection is established, both hosts can transmit TCP segments back and forth to each other. This is going perfectly if conditions of the network are optimal. The packets can be delayed, fragmented, corrupted, or lost (or a combination of those) due to network anomalies. Because TCP is a reliable protocol, it specifies mechanisms to detect and recover such losses. However, most of all, TCP prevents error-prone events. TCP is a reliable protocol mainly because every segment has to be acknowledged. This will be described in Section 2.2.5 in more detail. First, the sender buffers all data before transmission. When the amount of data reaches the maximum segment size, the segment will be sent. By this method, the amount of overhead is minimised and thus contributes to a higher throughput. The only exception to this, is the use of the urgent pointer or the *PSH*-flag. The push and urgent function determine a specific segment which requires high priority to be sent as fast as possible. A good example for the PUSH function is when working in a terminal through SSH. When a command is entered, you want it to be executed directly. The buffer is not necessarily filled, because a command is likely to be a small number of bytes. Therefor, the *PSH*-flag is set



(a) Sliding window concept

(b) Receivers window concept

Figure 2.7: Graphical representation of the sliding window effect ([2]fig.1 and fig.2)

and the command is transmitted. Consequently, after the segment is sent, instead of waiting for the acknowledgement of each segment before sending the next segment, the next segment will be sent. The number of segments can be sent is defined by the window size. When the first segment is acknowledged, the window slides and unlocks space for the next packets to be sent (see Figure 2.7a).

The size of the sliding window is directly determining the throughput of the TCP flow and depends on the round trip time. Making the window larger, will increase the throughput. However, a large window results in a higher probability of packet loss because the network and the receiver have resource limitations [62]. Therefore, the receiver advertises to the sender its resource availability by means of the receiver window as described in Section 2.2.2. Figure 2.7b illustrates this simple concept. The sender receives the changed value of the window and will not send more segments to prevent overflowing of the receiver's buffer.

2.2.5. Reliability

The TCP sender is responsible for the proper delivery of the packets to the receiver. The TCP standard defines the acknowledgements of segments by using sequence- and acknowledgement numbers [55]. Every segment has a sequence number attached to it. The receiver will send an ACK together with an acknowledgement number. This acknowledgement number does *cumulatively* acknowledge the segments. Therefore, this number indicates that all segments having a smaller sequence number have been delivered. Both the client and the server must keep track of the segments sent over the connection. To do this, the byte stream is categorised and assigned to pointers[40]. These pointers are stored in the TCB. Four categories are described to define the transmission state of the stream at the sender side.

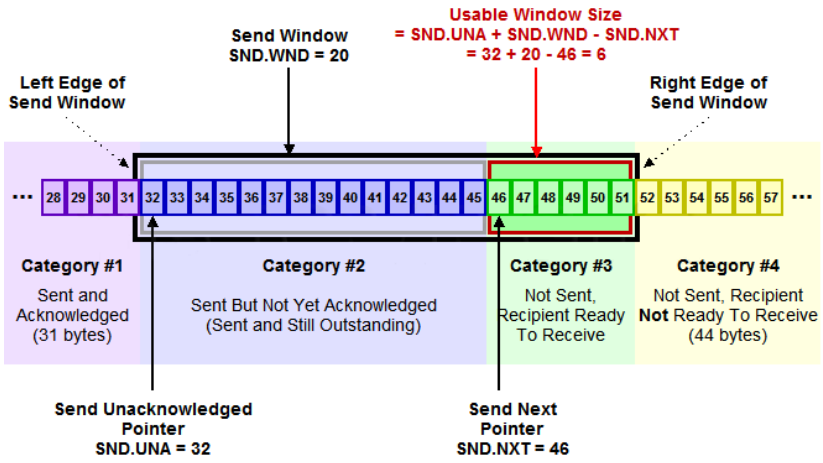
1. Bytes sent and acknowledged
2. Bytes sent but not yet acknowledged
3. Bytes not yet sent for which recipient is ready
4. Bytes not yet sent for which recipient is not ready

These categories are separated by the designated pointers. Three pointers are used for this.

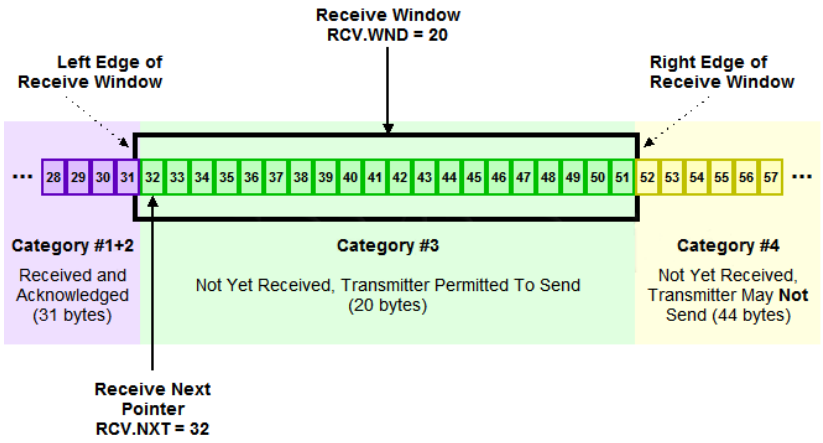
1. **Send Unacknowledged (SND.UNA)**
   This pointer contains the first sequence number of the byte which is already sent, but not yet acknowledged. This corresponds to the first bye of category 2 as described above and is shown as the first blue byte in Figure 2.8a.

2. **Send Next (SND.NXT)**
   The sequence number of the next byte of data to be sent to the other device. This is the first byte of category 3 and corresponds with creen from Figure 2.8a

3. **Send Window (SND.WND)**
   The sending window represents the total number of bytes which are allowed to be in-flight at any time. Which means The sending window covers both category 2 and 3 from the transmission state. As can be seen in Figure 2.8a, adding the sending window to SND.UNA will result in the first byte of category 4.



(a) TCP send categories and pointers



(b) TCP receive categories and pointers

Figure 2.8: Graphical representation of the transmission categories and pointers. Complementary categories for sender and recipient have the same colour. ([40]fig.219 and fig.220)

At the receiver side, also a transmission state for all bytes is being tracked. The receiver should send proper acknowledgements and should advertise its buffer space to the sender. The transmission states are divided in three categories for the recipient.

1. Bytes received and acknowledged. This is the complement of category 1 and 2 of the sender's side
2. Bytes not yet received for which recipient is ready
3. Bytes not yet received for which recipient is not yet ready.

These categories are separated using two pointers.

1. **Receive Next (RCV.NXT)**
   This pointer contains the sequence number of the next byte that is expected to arrive. This also means that the last acknowledgement number sent to the sender is RCV.NXT minus 1. RCV.NXt is inicated as the first byte of (green) category 2 as can be seen in Figure 2.8b.

2. **Receive Window (RCV.WND)**
   The receive window is used to let the sender know how many bytes this device is willing to accept. This corresponds usually to the allocated buffer space of the receivin device. In Figure 2.8b you will see addition of RCV.WND to RCV.NXT points to category 3.

Since the pointers of both the sender and receiver are complementary, both hosts know the state of transmission and both are in control of the flow. The sender keeps track of a usable window as shown in Figure 2.8a, which is used to prevent sending more bytes than the receiver is able to process. Conversely, the receiver can advertise its buffer space to notify the sender to adjust the window size. The SND and RCV pointers are updated each time data is exchanged. These values are extracted from and updated to the fields of the TCP segments.

- **Sequence Number** should identify the first byte of data being transmitted first. Therefore, this value should be equal to the SND.UNA pointer.
- **Acknowledgement Number** is set by the receiving side and identifies the next byte number to be expected. So the acknowledgement number is set to the value of the RCV.NXT at the receiver side. At the sender's side, receiving an acknowledgement number will update the SND.UNA pointer and the window can slide to unlock new room for the next bytes to be sent.
- **Window Size** is sent in each TCP segment. The window size is advertised by the receiving side and used by the receiving side. Therefore, the send window of one device is the receive window of the other device and vice versa.

This is how TCP devices manage the sliding window effect and handle the acknowledgement concept. However, this only works when every packet transmitted arrives correctly and in the right order. Due to any number of internet environment conditions, packets can get lost or arrive in a wrong order. TCP would not be a reliable protocol if it has no methods to detect and solve this issue.

Retransmissions
For each segment is sent, a retransmission timer is started for a predestined time. When the packet is not acknowledged before the retransmission timer times out (called a *Retransmission TimeOut (RTO)*, this segment is retransmitted. Because the acknowledgements are cumulative, there is no knowledge if only this segment is lost or all consecutive segments are also lost. So the TCP device does not know which segments are lost after this segment (except when using Selective Acknowledgement, see Appendix A). Two methods can be used for retransmission. First is to retransmit all segments after the lost segment. This is a more aggressive, pessimistic method. All segments will be recovered, but probably a lot of unnecessary retransmissions are being performed. Second method is to only retransmit the timed-out segments. This is a more conservative, optimistic method. Considering the other segments arrived correctly, this method will retransmit all segments very effectively. However, this method will cost a lot of time quickly when more consecutive segments are lost, because for each one a time of RTO passes. The RTO value should be chosen very precisely. A small RTO means a loss can be detected quickly and thus be fast recovered. However, if the RTO is too small, a premature loss will be detected and an unnecessary packet will be retransmitted [2, 40, 43]. This will be a waste of network resources and result in higher congestion. Ideally, the RTO is set to a slighter larger value than the *round-trip time (RTT)*. The RTT is, together with the window size, an important value for the throughput of the connection[43]. This requires a good estimation of the RTT, which is not even a constant value. The RTT depends on network parameters, like the distance between the hosts, other network traffic, hardware resources, etc. Many researches are done (and still being done) for estimating the RTT value.

A natural way of calculating the RTT is by noting the time a segment is sent, noting the time the acknowledgement receives and subtract those values. Because the RTT varies, an average value for RTT will neglect large deviations. In RFC2988, a smoothing function is stated for this calculation [40, 54]:

$$NewRTT = (\alpha \times OldRTT) + ((1-\alpha) \times NewestRTTMeasurement) \tag{2.1}$$

The α determines the aggressiveness of the smoothing function. Lower values can make change RTT more quickly, but can also cause overreaction. High values provide better smoothing, but react slow to changes in RTT measurements. However, using this method is still issued by the concept of *acknowledgement ambiguity*. When a segment is retransmitted, there is no distinction between the original and the retransmitted

segment. The acknowledgement could still be acknowledging the original segment and not the retransmitted one. Besides this, the acknowledging segment could also have been delayed. Phil Karn proposes a new solution by changing the sampling method for taking the most recent RTT measurement [35]. When a retransmission for a segment has been sent, the measurements for this RTT will be ignored. In addition to this, the Karn's algorithm uses a *RTO back-off* for more accurate RTT measurements which are not contaminated by acknowledgement ambiguity. The RTO will not change for a segment that has been sent more than once, and this same RTO will be kept for the next segment. Only when an acknowledgement without any retransmissions in between is received, the RTO will be renewed from the recalculated smoothed RTT[33, 35]. This allows a TCP device to respond with longer RTO's to occasional circumstances that cause delays to persist for a period of time on a connection, while eventually having the RTT settle back to a long-term average when normal conditions resume [40].

2.2.6. Congestion Avoidance

In Section 2.2.4, the concept of the sliding window was explained. The window size is related to the amount of data the receiver is able to process. The window will be reduced when the available buffer space is getting smaller and will be increased when more buffer space gets accessible. This method by which TCP implements flow control. Flow control is an essential part of TCP, as it is this method by which devices provide feedback of their status to each other. In optimal conditions, the window size matches the space left in the buffer after a segment is received and will not change over time. This will only work when the receiver can process the data in the buffer at the same speed as the packets are arriving. In a realistic environment, this is definitely not the case. The device might be dealing with a lot of other TCP connections at the same time and is very unlikely to process the data immediately. Furthermore, the application might not be ready for the data and backs off the data temporarily. When this happens, the receiver must notify the sender to transmit packets at a lower rate, to prevent overflowing the buffer. The receiver will send a smaller window size in the next acknowledging segment.

However, what happens when the receiver cannot process any more data. The window will be reduced at every acknowledgement until there is no window left and has size zero. The sender will not be able to send data anymore. When the receiver has processed some part of its buffer, it can reopen the connection by increasing the window. However, opening with a window too small, the performance will be reduced due to the large overhead. The adjustments of the window should be changed very carefully, otherwise serious issues will occur in the operation of TCP. There is an important difference between *reducing* the window and in *shrinking* the window for TCP operations. The window has two edges. The left edge is positioned at the first unacknowledged byte (see Figure 2.7). The right edge of the window is positioned at a distance the size of the window size away from the left edge. *Reducing* the window size is making this distance shorter, by adjusting the window size in the header field. When the window size is reduced with the same size as the segment received, this will result in the right edge "freezing". There is no room for extra bytes to be received. It could be even worse when the window size is reduced more than bytes which are already in transit. Then the right edge will move to the left and this is called *shrinking* the window. An example of this is shown in Figure 2.9 [40]. This effect will not break down the TCP connection, but has a great impact on the efficiency. Because this new window size will take back the usable window on the sender side to a value which is lower than SND.NXT. In other words, this will force already sent data to be retransmitted unnecessary. This is very inefficient and will take a worthless part of the network load. To prevent entering this state, TCP devices are not allowed to shrink the window. The receiver has to be patient and reduce the window to a size that the right edge is frozen.

Another problem concerning the window size is when the window is closed by the receiver. The window size is zero and the sender is not allowed to send any data. The sender cannot send anything until it receives an acknowledgement from the receiver with an increased window size. The problem with this is that the acknowledgement will only be sent as a reply to the sender, but the sender is not allowed to send anything. When the sender does not receive anything it can conclude the connection is lost and to terminate the session. To prevent this from happening, the sender will send *probe* segments regularly. These probes are segments containing no bytes of data and prompts the current window size. The sender will continue sending these probes until a segment containing a non-zero window size is received. When the receiver is ready to receive the next segment, it decides to reopen the window. Then the next issue arises. When this new window is chosen too small, this leads to the generation of many small segments, which will congest the network with a lot of overhead.

The sizes of TCP segments are determined by the maximum segment size (MSS). The segments should

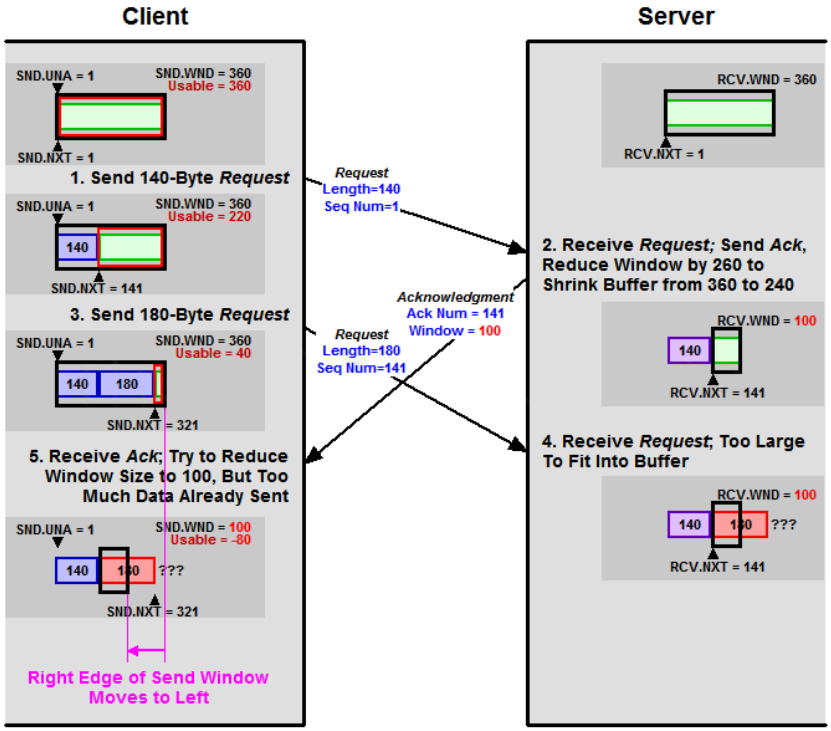


Figure 2.9: Graphical representation of shrinking window effect ([40] fig. 227)

not be too large or a risk of having them fragmented in the IP layer occurs. When the segments are too small, the performance of the connection is greatly reduced. The MSS parameter ensures that TCP is not allowed to send any segment larger than the MSS. But the sliding window mechanism does not provide any minimum size of the segments. This could have problems when having a very small window size; a window size even smaller than one MSS. Figure 2.10 shows an example of when this phenomenon happens. The receiver's speed of processing the incoming bytes is slower than the rate of the bytes being transmitted. The sender has a huge amount of data to send and transmits all bytes as fast as possible. Each time the window size is reduced by a small amount by the receiver and the sender will send a full window of data. This results in getting a very small window size or even a temporary closure of the window. This is not a failure of the connection, because the data is still transmitted properly. However, this is done with a very low transmission rate. This phenomenon is called *silly window syndrome (SWS)*. Several algorithms for SWS avoidance have been developed [15, 40].

Both the sender and receiver contribute to the SWS effect. First look at the receivers contributions. The receiver is reducing the window size to smaller and smaller values, which causes the right edge of the window to move with very small steps. To avoid SWS, it is not allowed to do such small reductions of the window size at the receiver. The receiver is restricted from moving the right edge too small. The minimum is 1 MSS or one-half the buffer size, whichever is less. Therefore, the window at the sender's side will slide with at least this value or else the window is closed. Secondly, the SWS avoidance is executed by adjusting when the sender is allowed to send new bytes. The method is invented by John Nagle [52]. Two rules apply for when the sender wants to transmit[2]:

1. As long as there is no unacknowledged data outstanding on the connection, the application can send its data as soon as it wants.

2. While there is unacknowledged data on the connection, the sender will have to store the data in the transmission buffer. Until all outstanding data is acknowledged, or if the transmission buffer reaches the size of one MSS, the segment is transmitted. This even applies when a PSH-flag is set.

Algorithms

By adjusting the window size, both devices ensure a sending rate that matches the speed the recipient can

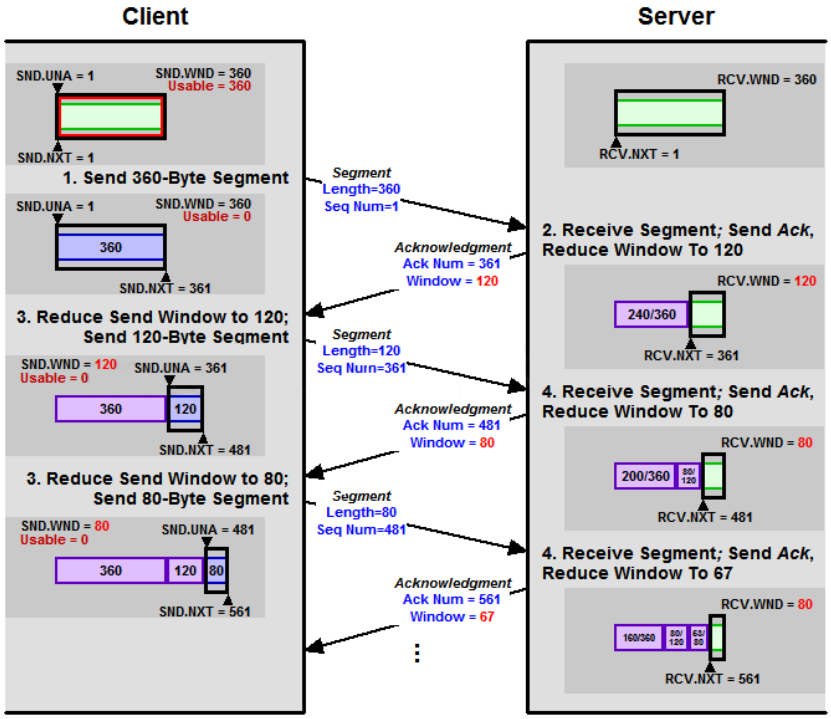


Figure 2.10: Graphical representation of Silly Window Syndrome ([40] fig. 228)

handle the packets. In optimal conditions the sender is transmitting packets at the maximum rate. The capacity is limited by the transmission link. An schematic representation is shown in Figure 2.11 [34, 50]. The link between the sender and receiver is denoted as the bottleneck. Vertical dimension is the transmission rate (bit/s) and the horizontal dimension represents the time, such that the area presents the packet size in bits. P_b represents the spacing between two packets on the bottleneck link if the sender is sending at maximum capacity. At arrival at the receiver, the time between the packets does not change, hence $P_b = P_r$. Assuming for each segment an acknowledgement is created at the same speed than the time between two acknowledgements are sent is the same. Thus $A_r = P_r$, assuming each acknowledgement is smaller than the segment it acknowledges. Therefore, the acknowledgements will reach the sender in the same interval. Concluding that $A_s = A_r = P_r = P_s$. T

This concept explains why TCP is called a "self-clocking" protocol [50]. When enough packets are sent, this self-clocking system will automatically adjust to the available capacity and the connection will be in a transmission state equilibrium. However, the only problem is to start this system, because data gets flowing if acknowledgements are received to clock the packets, while the acknowledgements for their turn need packets to flow already. Jacobson developed an algorithm to start the "clock"[34].

TCP's sliding window system manages the flow control for a connection, only taking in account the settings of the devices. It lacks the opportunity to adjust the transmission rate based on fluctuations inside the network. It is common knowledge for flow-control systems that if an offered load in an uncontrolled distributed sharing system exceeds the total system capacity, the effectiveness decreases (even at times to zero) [27]. When this effect occurs for TCP it is known as a *congestion collapse*[2, 23, 34, 51]. The ***slow start*** is introduced in one of the first adoptions, developed by Jacobson. To tackle the problem of congestion avoidance, a *congestion window (cwnd)* is used at the sender side. The usable window is determined by the minimum of the congestion window and the advertised window. The slow-start algorithm starts by setting the congestion window to a minimum of 1 MSS. At each acknowledgment received at sender's side, the congestion window is increased with one MSS. This results in doubling the size of the congestion window at each RTT. This method lets the sender explore the transmission rate limit of the network with exponential speed. In addition to this a ***congestion avoidance*** algorithm is applied. This algorithm will approach the network limit with a linear speed. Each time an acknowledgement is received the congestion window is increased by $1/cwnd$ (in units of MSS), resulting in an increase of exactly one MSS every RTT. The sender sends each RTT an amount of one

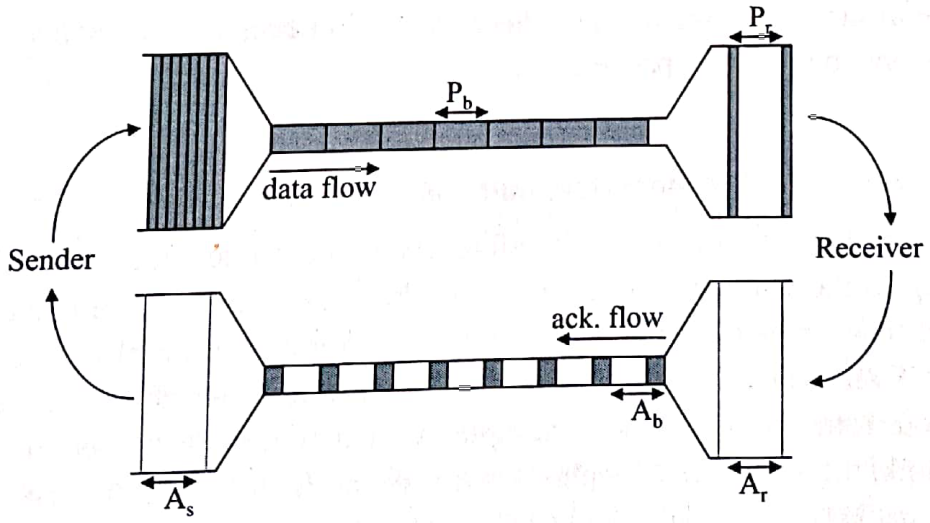


Figure 2.11: Explanation of TCP self clocking property [50](fig.5.8)

$cwnd$, so increasing the congestion window by $1/cwnd$ results in a linear increase. The slow-start algorithm continues until the congestion window reaches the value of the *slow-start treshold (ssthresh)*. The value of the $ssthresh$ depends on the congestion of the network. After a loss is detected, the congestion window is set back to 1 MSS and will restart the slow-start phase.

TCP Tahoe implements both the slow-start and the congestion avoidance algorithm[44]. Also Jacobsen added a ***fast retransmit*** algorithm to TCP Tahoe [34]. The fast retransmit enhances the packet loss detection. When a packet is lost, no acknowledgement will be received. But the segments which are transmitted correctly after the lost segment will send a acknowledgement. Because of the cumulative acknowledgements, these acknowledgement has the same number as from the previous acknowledgement. It misses some sequence numbers in between, which means only it can only acknowledge the last cumulative received number. So receiving duplicate acknowlegdements (double acknowledgement numbers from consecutive segments) is an indication to a packet being lost. After three duplicate acknowledgements, the sender decides the packet is lost. Now when this happens, the congestion window is reduced to 1 MSS and the lost segment is retransmitted. Only costs too much of the performance of the TCP connection. *TCP Reno* proposes a new solution when a segment is retransmitted after a triple acknowledgement is detected [2, 3]. The ***fast recovery*** algorithm is introduced. Instead of going into slow-start phase, the congestion window is halved. Having received multiple acknowledgements is an indication some of the segments after the lost segment did arrived. So the transmission rate should not be broken down to a bare minimum and can be recovered more quickly. In Figure 2.12 an overview is of the different phases the algorithm goes through. The efficiency of an algorithm is denoted by the ratio of the gray area and the area under the network limit.

The fast recovery algorithm enhanced the efficiency of the TCP connection. But still some minor issues occur when in a small period more than one segment are lost. For each lost segment the congestion window is halved. So when a new lost segment is detected, before the connection was recovered the windowsize is reducing exponentially. Floyd et al. added a simple refinement to TCP reno's fast recovery, transforming it into *TCP NewReno's* algorithm [24]. A distinction is made for a partial acknowledgement and a new data acknowledgement. Before retransmitting the lost segment, which is discovered after three duplicate acknowledgements, the highest sequence number which is already sent is noted. A new data acknowledgement is received for the the highest sequence number transmitted. A partial acknowledgement acknowledges only the recovery of the first error and indicates more segments are lost. The congestion window is halved after first transmission. For each duplicate acknowledgement the congestion window is increased by 1 MSS, because this indicates a segment has arrived correctly. A partial acknowledgement reduces the window by 1 MSS and retranmsits the newly detected lost segment (so usable window does not change). An overview of the operations of the TCP NewReno's fast recovery is shown in Figure 2.13. Over the years many algorithms have been developed. Mainly focusing on three problems to tackle:

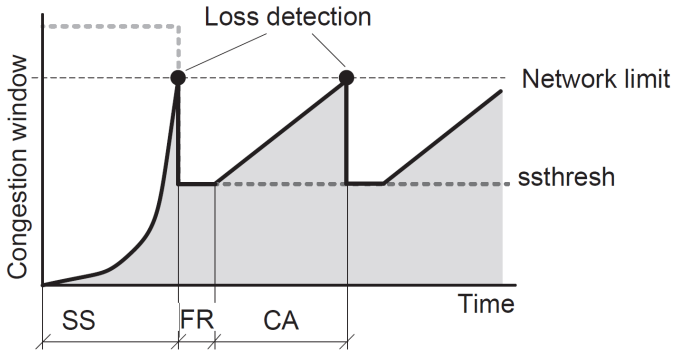


Figure 2.12: Congestion window dynamics of TCP Reno ([2]fig.15)

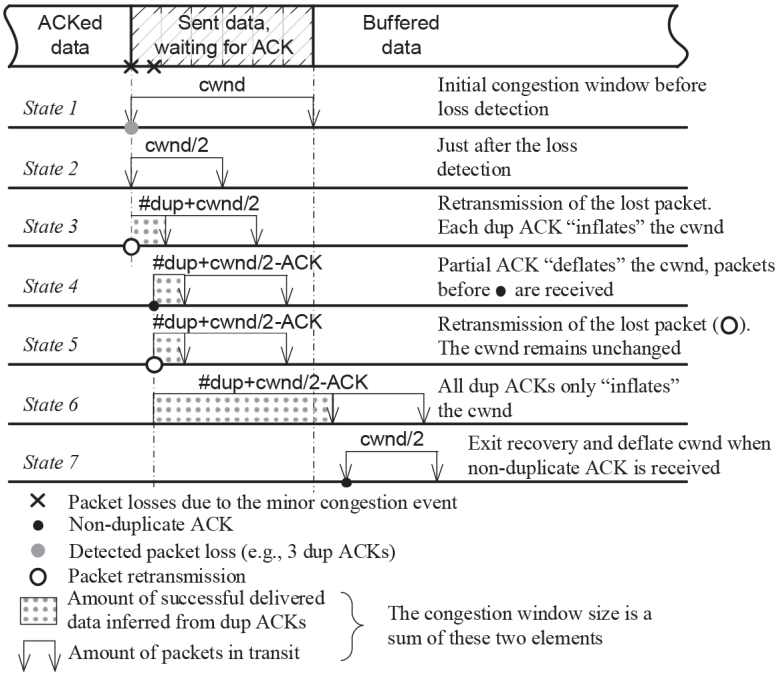


Figure 2.13: Explanation of TCP NewReno's Fast Recovery ([2] fig. 18.)

1. Improving RTO estimates
2. Enhancing detection of packets losses
3. Improving congestion avoidance algorithms

Currently the most-used congestion algorithms for TCP are NewReno and CUBIC [2]. TCP CUBIC originates from NewReno, with many iterations and other algorithms in between. CUBIC is designed for high-speed networks and improves fairness which is described below. Both algorithms progressed in improving packet loss detection and congestion avoidance. Other types of algorithms responds on changing RTT values, while NewReno reacts on (detection of) losing packets in the network.

Congestion algorithms are not only to find an transmission equilibrium for one connection on the network. A lot of traffic is using the same network resources. So the algorithm should take a fair share of the network load. It should not be expelled by the other network, but it should also not be the dominant connection and taking all resources. Chiu and Jain [13] developed an fairness index for this:

$$F = \frac{(\sum_i^n f_i)^2}{n \cdot \sum_i^n f_i^2} \tag{2.2}$$

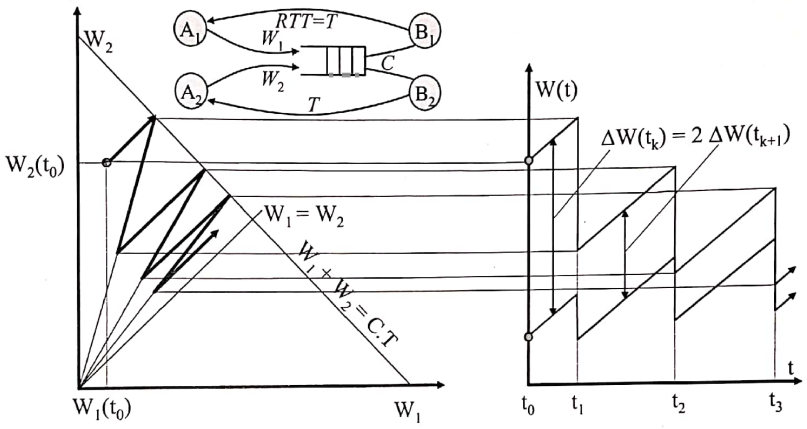


Figure 2.14: The trajectories of two congestion windows converging to the same share of network load ([50] fig.5.19

where n is the number of users and f_i is the part share of the i^{th} user. This index indicates the fairness or friendliness of an TCP algorithm. The range is from 0 to 1, where 1 is the highest achievable where every flow has the exact same share of the network. This is due to the additive increase an multiplicative (AIMD) behaviour of this method. The AIMD method is one of the most effective ways to approach this equilibrium[27, 69]. An example of the trajectory of this is shown in Figure 2.14. Two data flows are competing for their share of the network. Over time the congestion windows of both flows converge to almost the same size, which means they will use an equal share of the network resources.

2.3. Conclusion

Based on the problem statement of this thesis: "To what extent is it possible to implement a network diode on an FPGA under realistic network environments, using the Transmission Control Protocol", we researched the current diode products on the market, which was described in Section 2.1. From the developments in this industry, we learned the bottleneck of this project. TCP is a bidirectional flow, which has to be converted to a unidirectional flow. The standard solutions use PC's, which run proxy software to solve this issue. The only hardware implementation is the one-way diode connection, which does not cover the features of the TCP. The data flow is converted from bidirectional to unidirectional at the sending side of the diode. The information is sent using a custom-made protocol designed for the user. At the receiving side, the information is converted back to the bidirectional TCP flow. This has the advantage of an adjustable diode protocol. The functionality can be modified by updating the proxy PC's. Although the disadvantage of this is the costs of the diode and the maintenance of those proxy PC's.

From the operating mechanisms of TCP described in Section 2.2 we can conclude it is a very reliable protocol. It has many features to establish the connection, manage the connection while taking in account other traffic and has enough room to implement new features for improvement. But those characteristics for TCP are only valid when communication is available both ways. A bi-directional connection is vital to the TCP methodology. The sending side of the connection requires feedback from the receiving side to operate. There is no adaption of TCP with only one-way traffic allowed. TCP implements algorithms to adept to the current state of the network, which is mainly detected by packets loss or fluctuations. those algorithms provide heuristic solutions to recover the connection, while achieving high efficiency and taking in account a fair share of the network resources. With this knowledge we continue to find a method to define a simplified model of our system as described in Section 1.4 for analysis.

3

System evaluation

With the knowledge described in Chapter 2 we gained more insights to develop a model for evaluation of the system. From Section 2.1 we know the bottleneck of designing the network diode for using TCP is the conversion from bidirectional flow to unidirectional flow. The common solution is to process this in a proxy PC. The goal is to perform this conversion in hardware on the same device where the diode is implemented. A new method should be developed for this conversion and, therefore, the concepts of TCP was researched in Section 2.2. The TCP protocol was designed to provide feedback of the status of the hosts and network. The protocol can change settings to adapt to these anomalies. It can manage and control data flow and has features to improve reliability. Not only does it manage the traffic of the connection but it also distributes the network load for other traffic. To design a method for converting TCP data flow, first the behaviour of these kind of traffic has to be analysed.

In this chapter, a method for evaluation of the system is developed. In Section 3.1, an approach for a simplified model of the system is presented. To satisfy goal 1 (Section 1.3; Design requirements: minimise resource utilisation, maintain a reliable connection and maximise data flow), the most important factors are determined. To determine the influence of these factors, we will investigate the relationship between the throughput of both networks and the buffer size of the diode. To find a solution for this relationship, the methods for calculations on throughput prediction are discussed. This results in Equation (3.6), a general formula for determining a throughput of a connection using TCP. Thereafter, a division is created for settings which can be controlled and to what parameters will affect the network on both sides.

To analyse this set of a parameters, a conventional simulation software is selected in Section 3.2. Research shows three common used programs for these kind of simulations: MiniNet, OmNet++ and NS3. For each of these programs, the suitable features for our system are evaluated and presented in a clear trade-off table. Finally, a selection is made which will be discussed in more detail in Chapter 4.

3.1. System Analysis

The data diode will be inserted in the connection link between two domains. One domain is a trusted, highly secured domain. The other is a less secure domain. Under no circumstances there should be any kind of flow from the high-secured domain to the low-secured domain. For simplification, the **high-secured (trusted) domain** will be referred as the **red domain** and the **low-secured domain** will be referred as the **black domain**. In Section 1.3, the second goal of this project is to develop a model for system evaluation. To start with simplifying the system, we state that the system consists of two parts. The data diode splits the system in two parts, the black network and the red network. The only common element is the information flow from the black network to the red network, because all the other flows are blocked by the diode. In Section 2.2, it was stated that TCP cannot operate with an unidirectional connection. Therefor, the TCP stream has to be handled at both sides of the diode. It is up to this project to determine what parts of the TCP are being kept when information is passed through the diode. Consequently, a protocol for the data transferred by the diode should be defined. The TCP streams on each side are operating independently of each other as there is no method to supply feedback about the status of each network to the other side. This rule must

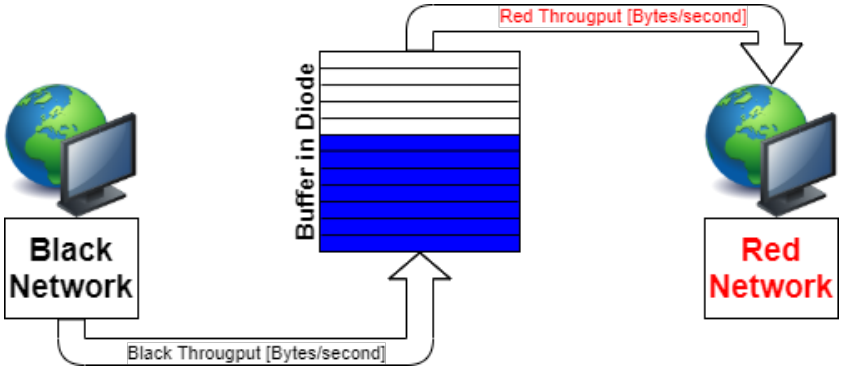


Figure 3.1: Simplified overview of system

apply, otherwise the data diode does not satisfy its primary function: establishing a one-direction gateway and blocking all kinds of traffic the reverse way.

To this extent, a simplified model can be formed as is depicted in Figure 3.1. The information transferred through the diode is the only piece of shared data. This data has to be temporarily stored in a buffer. The data diode should be established on an FPGA, which does not have an unlimited buffer size. Resources on FPGA's are expensive and these should be minimised as was stated in Section 1.3. The TCP stream is going to flow from the black network to the diode. The black part of the diode will handle the handshake and acknowledgements to the sender from the black part as well as it transfers the payload over the diode. There will be a stand-alone TCP stream between the source in the black network and the receiving side of the diode. The imperative information will be added to the buffer inside the diode. The sending side of the diode, connected on the red network, will empty this buffer by setting up a new TCP stream at the red side. The TCP stream at the black side will fill the buffer and the red side will empty this buffer.

3.1.1. Functional Requirements

From the model as was described in the previous section and looking at the depicted overview in Figure 3.1, the question arises: To what extent is the buffer size affected by the throughput of both networks? And what minimum buffer space is necessary? The amount of data stored in the buffer depends on the behaviour of both parts of the network. The black network will send data to the diode and can only negotiate with the receiving side of the diode. There is no possible method for the black source to adjust its sending rate to any anomalies at the red side of the network. Vice versa is it not possible of the red network to notify the black network about the status of the TCP stream or the amount of buffered data. The sending rate of a TCP stream is defined by its throughput. The size of the data in the buffer can be calculated by the difference of the throughput of the black network and the throughput of the red network. Only this is not all of the data to be stored in the buffer. TCP is a reliable protocol and has mechanisms to detect and recover lost packets as was explained in Section 2.2. The sender in the TCP stream must be able to retransmit any lost segment. The segment have to be stored until it is acknowledged by the receiver. All segments which are already sent but not yet acknowledged (see category 2 of sending transmission state in Section 2.2.5), should be stored in the buffer. The maximum amount of data to be sent, but not acknowledged, is the size of the sending window in the red network. A comparison could be made with routers, which act as an in-between device of a connection. Routers connect multiple devices and also have a temporary buffer to transfer data using TCP. The rule-of-thumb for routers is to have a buffer of size $B = \overline{RTT} \times C$, where $\overline{RTT}$ is the average RTT and C is the maximum capacity of the connection link [6]. This set of rules can be contracted to the following formula for calculating the buffer size dependent on the throughput of both networks:

$$Buffer \approx Window_{Red} + (Throughput_{Black} - Throughput_{Red}) \tag{3.1}$$

From Equation (3.1), we can conclude one relationship between the black and the red throughput of the system. The buffer size does need a maximum value, it is not possible to keep increasing the amount of data to be buffered. To prevent the buffer from overflowing and ever increasing, the system should apply to the following rule:

$$Throughput_{Black} \leq Throughput_{Red} \tag{3.2}$$

The black network should at least process the segments at the same speed as the red network is supplying new data, otherwise the buffered data keeps increasing. This sets two new goals for the system:

1. Determine the maximum capacity of the red network with regards to the throughput of the TCP stream
2. Define a method to control and set a limit to the maximum throughput of the black network

To determine what size of buffer space is required, the throughput of the black network should be limited and the behaviour of the throughput of the red network should be predicted in advance. There is absolutely no method to provide feedback of the status of the red network to the black network without breaking the main function of the data diode. The black side is not even allowed to know the saturation level of the buffer at any moment, as this could leak information of the red network. We must ensure to let the throughput of the black stream not to exceed the red network. The black throughput should be limited to the maximum value of the red throughput. We need to find out how the throughput of a TCP stream can be determined and to what extend it can be controlled.

3.1.2. Throughput calculation

To determine the maximum throughput of the red network and to control the throughput of the black network, we need to find a method to evaluate this. In Section 2.2 was explained, TCP has many mechanisms to maintain its reliability and takes care of a smooth use of network resources. It implements a flow control to prevent errors at the receiver side, while simultaneously it takes in account other traffic using the same network resources to avoid congestion. That last part allows the most opportunities for improvement as this has been kept changing over the years. The huge increase in internet traffic over the past decades did not only introduce new problems regarding the amount of data to be transferred, but many more changes were made to be taken in account. The type of data, the type of protocols, the network architecture, the physical devices transporting internet traffic, the dependency on real-time data, the distance between devices, wireless and local networks and much more changed over time [2, 65]. This requires all-time changing new tactics and strategies for the Transmission Control Protocol. As was stated in Section 2.2, TCP aims to the optimal method to transfer data with the highest efficiency. The efficiency is improved by reducing the amount of retransmits. To reduce retransmissions, lost segments should be detected and identified as soon as possible. Consequently should these lost segments be sent in a method to recovery the TCP stream without losing as less of data flow as possible. In addition, the loss of packets could be prevented by avoiding congestion, which will improve the efficiency of the throughput.

These mechanisms are updated and renewed by proposing new congestion algorithms and adding features to TCP in the options field. There is not a standard method to calculate or determine the expected throughput of a TCP stream. To look at the effects of the congestion algorithm, we are going to evaluate the algorithm TCP NewReno as an example. Congestion algorithms introduces a congestion window as explained in Section 2.2.6. The used window is the minimum of the advertised window of the receiver and the congestion window. The size of the congestion window is dependent on the algorithm. TCP NewReno has defined four different phases for the congestion window: slow-start, congestion avoidance, fast retransmit and fast recovery. Fast recovery is actually more of a modification to the fast retransmit as defined in the TCP Reno algorithm. Plotting the theoretically congestion window over time for TCP NewReno would look something like Figure 3.2.

The graph depicted in Figure 3.2, represents the congestion window over time for the TCP NewReno algorithm [50]. Some simplifications and assumptions are considered for this representation. The maximum window size is denoted by W and is represented in units of packets, assuming all packets have the same size. Furthermore, RTT is assumed to remain constant and the window size will remain constant during one RTT. The transmission rate would then be W/RTT [packets/s]. As is depicted in Figure 3.2, the transmission behaves periodically after the first retransmit triggered by an RTO. The window is reduced to half its size and the algorithm starts the congestion avoidance (CA) phase. In the CA phase, the window size increases by one packet size every RTT. The slope a, is a linear increase with value $1/T$. During a periodic cycle D, the window increases from $W/2$ to W. Therefore, the periodic cycle will take $\frac{W}{2}RTT$ seconds. One cycle D is exactly $\frac{D}{RTT}$ number of RTT's. The amount of packets during one cycle can be calculated by taking the area under one of the saw-tooth's depicted in Figure 3.2. The number of packets M will then be [50]:

$$M = \frac{D}{RTT}\frac{W}{2} + \frac{D}{RTT}\frac{\frac{W}{2}}{2} = \frac{3}{8}W^2 \tag{3.3}$$

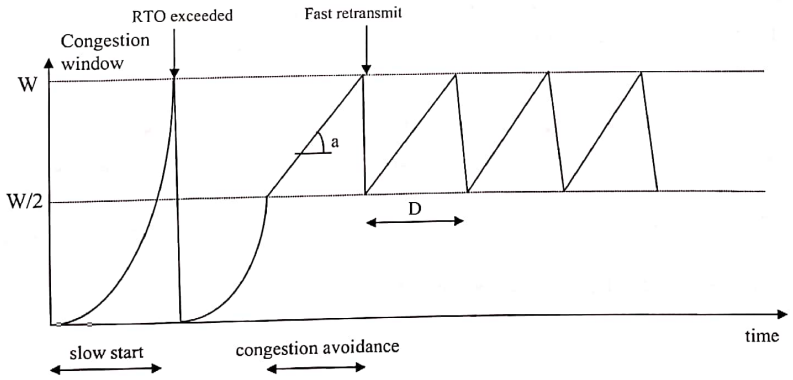


Figure 3.2: The congestion window as a function over time, assuming the RTT remains constant ([50] fig5.17)

This is, with the assumption only one packet is lost when the fast retransmit is triggered. Considering M packets are being transferred during one periodic cycle D, the loss probability p equals $p = 1/M$. Therefore, the maximum window size becomes $W = \sqrt{\frac{8}{3p}}$. The average transmission rate, calculated over the periodic cycles will now be [50]:

$$R = \frac{M}{D} = \frac{3W}{4RTT} = \frac{\sqrt{\frac{3}{2}}}{RTT\sqrt{p}} \tag{3.4}$$

This simplified analysis led us to Equation (3.4). Evaluating this equation, will lead to some important conclusions. First of all, the throughput of TCP stream is inverse proportional to the RTT of the connection. TCP stream's with a small RTT will react more aggressive to other traffic with regards to the network share. As was explained in Section 2.2.6, TCP protocols are evaluated on their friendliness towards other streams. TCP streams with smaller RTT will take a larger share of the network load as streams with a larger RTT. Secondly, the throughput of a TCP connection is inverse proportional to the square root of the loss probability. Therefore, Equation (3.4) is termed the *inverse square root law* for TCP streams throughput [50].

Over the past decades the number TCP CA algorithms increased to satisfy the new infrastructures of modern networks. These evolutionary process of new algorithms could be categorised in three strategies [2]:

1. **Reactive algorithms**
   These algorithms adjust the window size when packet loss is detected. Their focus is on improving methods for detecting packet losses earlier and to recover the stream with highest efficiency.

2. **Proactive algorithms**
   Proactive algorithms enhanced methods for RTT estimations. By detecting alterations in the RTT, delays can be detected. Good delay estimations can predict the current maximum network capacity and this algorithms uses this calculations to adjust the window size.

3. **Reactive algorithms (inc. bandwidth estimation)**
   These algorithms are a slight modification to the standard reactive algorithms. The window size is still only adjusted when a loss is detected. But the reduction value changes based on the current bandwidth estimation. These algorithms are common to be used for wireless networks, where the network capacity fluctuates wildly [2, 45].

The focus of most enhancements of these algorithms was to improve the performance for modern networks with high bandwidth. The main theme in this line of development is the inverse square root law. The window size is controlled by the additive increase multiplicative decrease (AIMD) algorithm, which will cause this relationship. New algorithms proposing methods using multiplicative increase multiplicative decrease (MIMD) schemes, show better results with regards to proportional fairness [37]. Finally, algorithms could be analysed by a general increase multiplicative decrease (GIMD) scheme. This results in a general relationship between the throughput (ω) and the loss rate (α) [48, 53]:

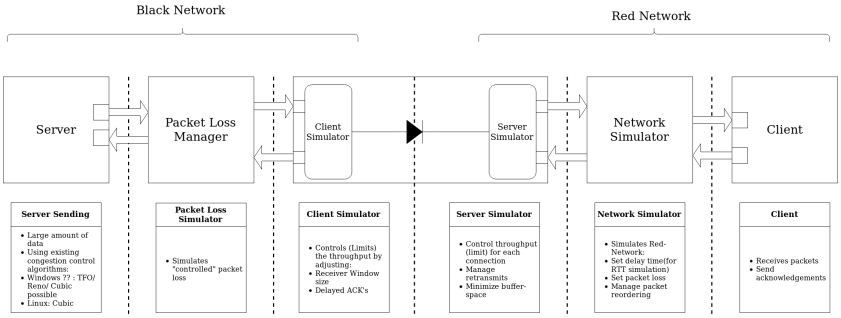


Figure 3.3: Overview of the parameters affected by each device in their part of the network

$$\omega \approx K\alpha^{-b} \tag{3.5}$$

In Equation (3.5), K is a constant dependent on which TCP CA algorithm is used, but also on which testing models are used [48]. K is dominantly dependent to the decrease rate of the window size in the algorithm and b is strongly dependent on the increase rate of the window size. For TCP NewReno, the b is equal to $1/2$, which leads to the inverse square root law. Scalable TCP proposed the MIMD method and will change the value of b to 1 [37]. Furthermore, in most studies, the rate at which losses occur are assumed to be linear proportional with the rate window size changes [47]. In addition to this, other studies show much more situations where the rate loss is independent on the algorithm [4]. Moreover, losses could occur in clumps or batches which will have different behaviour for the throughput. Finally, researches also have shown that the loss rate is dependent on the throughput, which means they are dependent on each other.

Concluding the findings from researches for throughput determination, the most important factor is the algorithm for adjusting the window size. The packet loss probability is coherent dependent on the throughput and vice versa. From Equation (3.5) we can conclude the algorithm affects the throughput directly and determines the aggressiveness level of reaction to packet losses. When the TCP stream reacts to aggressive, it comes at the expense of the TCP-friendliness and will take an unfair share of the network load. Because we want to predict the throughput of the red network, but also want to limit the throughput of the black network, we need a formula with parameters which can be verified or controlled. The general calculation for the throughput, without using any algorithms, would be equal to W/RTT. Combining this together with Equation (3.5) from Maulik and Zwart[48], we propose the following formula:

$$Throughput \approx \frac{WindowSize}{RoundTripTime} \times C \times p^{-b} \quad \text{[Bytes/s]} \tag{3.6}$$

In Equation (3.6), C and b are dependent on the congestion algorithm and p is the packet loss rate. This equation can figure as guide for determining the configurations for both networks. From each variable in Equation (3.6) we are going to dissect if these are controllable or dependent for each side of the diode in the next section.

3.1.3. Black Throughput

The system is separated in two networks. Each of them operates independent of the other, except the information flow. The red network has to wait for the information coming from the other side of the diode. That is why the red network can not send at a higher rate as the black network and also why the black network is not allowed to send any faster than the red network is capable of as was explained in previous section. The functionality of the system can be divided over 6 different devices as is depicted in Figure 3.3. Each side has 3 devices, mainly doing the same task. For the black network, there is a source available which transfers data to the diode. At the diode a is process running, which accepts the packets from source and handles them by sending acknowledgements and transfer them over the diode. In between the black network's source and the black part of the diode, a packet loss manager is installed for simulation purposes. The packet loss manager will simulate packets getting lost due to errors in the network which is not really there. The red part is a copy of the black part except the position of the source and destination are swapped. The red part of the diode is responsible for transferring the data over a TCP stream.

From Figure 3.3 we can extract a set of parameters which will have an effect on the variables in Equation (3.6). We are going to discuss these parameters for each side of the diode. Starting with the black part

of the network, the parameters are being distinguished by means of the controllability or the dependency by the diode part of that side. The diode part in the black network is the receiving side of the TCP stream. This part should limit the throughput as was explained in Section 3.1.1.

Controllable parameters
The black part of the diode must function as the receiving side of the TCP stream. It should negotiate the TCP options, acknowledge the incoming segments and transfer the data over the diode. To prevent the red part from failing due to an overload, it should limit the throughput of the connection. The receiver of a TCP stream has the following options to limit the throughput:

- **Changing window size**
  The TCP stream uses the sliding window mechanism for flow control. Commonly this is based on a buffer at the receiver's side. Only in this system the receiver is not capable of knowing the size and saturation of the buffer. Although the size could be fixed and be known to the black part of the diode, the saturation over time can still not be known. From Equation (3.6), we can see the direct effect on the throughput of the TCP stream. Reducing the window size will lower the sending rate of the source and results in a lower throughput.

- **Artificial delayed acknowledgements**
  The source of the TCP stream keeps track of which segments arrived successfully and which not. It calculates how many segments can be send until the amount of packets sent but not yet acknowledged is the same as the window size (see Section 2.2.5). Looking at Equation (3.6), what actually happens when the acknowledgements are sent with a delay, is that the round trip time gets larger. A higher RTT results in a lower throughput.

- **Artificial packet loss**
  In addition to these options, the receiver can decide to not send an acknowledgement for the segment and faking a loss of the packet. This will result in a higher loss rate and will lower the throughput following Equation (3.6).

Depending parameters
The black part of the diode is passive with regards to sending segments. It waits and listen to incoming packets and will react with a proper response. Therefore, this part of the diode is limited in its options to control the throughput of the black TCP stream. It is dependent on the segments transmitted by the source and to what happens to these segments during transmission. It is preferable if we could create an accurate prediction of the behaviour of the TCP stream. To determine the throughput of the black side, the following parameters should be considered:

- **Sending rate of source**
  The source will send packets at a certain rate to the receiver. The source can not send faster than its maximum capacity, but has to adept to the feedback about the network. The receiver can only notify the source to slow down by means of the methods described before.

- **Packet loss in black network**
  After the segments are transmitted by the sender, the segments are not guaranteed to arrive correctly. The segments can get corrupted, fragmented incorrectly, lost, arrive in incorrect order, etc. The packet loss rate is a deciding factor for the throughput, as we can conclude from Equation (3.6). An error in transferring a segment can occur due to plenty of causes. We highlight the most important error-prone network anomalies.

  - **Black network architecture**
    The architecture of the black network describes the infrastructure for the connected devices. The infrastructure provides the connections and links between devices. Network architectures exists in many shapes and feature many functionalities. All devices could be connected to each other, or all connections are linked to one device. The architecture determines the complexity of the routes and the saturation of the network resources. Consequently is the RTT affected by the architecture.
  - **Number of devices in black network**
    When more devices in are connected to a network, the complexity for routability increases and

consequently, requires a decent architecture. These devices will, presumably, be a heavier load to the current network resources and the risk for congestion increases.

  - **Number of hops from source to diode**
    Transferring segments from host A to host B is commonly routed via other devices. Each device a segment passes, is called a hop for internet packets. For example, a hop could be executed in a router, a switch or a hub. Each hop the segment uses, is an extra risk to an error in the segment.
  - **Other traffic on black network**
    Internet packets will share many (inter-)connections between devices. If a lot of devices are using the same resources, the risk on congestion increases. Packets can get delayed, lost or corrupted due to network congestion.

- **The congestion algorithm**
  The source is configured to use a congestion avoidance algorithm. From Equation (3.6) we can see this will directly affect the throughput. Not only the sending rate is affected by a factor but the used CA algorithm is strongly influential to the packet loss rate. A lot of CA algorithms already exist and the amount keeps expanding. To know what algorithms could be expected, we researched the most used algorithms for common network environments. TCP NewReno has been the default CA algorithm for many years since its introduction. Today, TCP Cubic is one of the most used algorithms for standard hosts. Three main OS distributors have adopted this algorithm in their network interfaces. TCP Cubic is the default in Linux kernels (since 2006, version 2.6.19)[8], MacOS (since 2014, OS X Yosemite)[22] and Windows (since 2017, version 10.1709) [9]. TCP Cubic is designed for high-speed networks with a high bandwidth-delay product. It focuses on heterogeneous networks, where the RTT and packet loss rate vary significantly. The algorithm shows good performance and great fairness properties when measured in real-world environments [2]. TCP Cubic is able to switch the increase for the congestion window from logarithmic to linear, based on a packet loss rate profile. The algorithm uses a formula with a constant C and a coefficient β. TCP Cubic is not able use the maximum available network capacity. Research on the steady state throughput of TCP Cubic shows the maximum is approaching 90%[10]. This number is strongly determining the factor C of Equation (3.6). The value b of this equation is dependent on the coefficient β of the TCP Cubic algorithm.

3.1.4. Red Throughput

The red part of the diode acts as the sender of the TCP stream. It will start to negotiate with the receiver and will establish the connection. This process is in control of sending segments and responsible for detecting packet losses and retransmit those. The red part of the diode is only dependent on the supply of data from the red network. Nevertheless, this does not cause any issues, because the red TCP stream does not get loaded when there is no data to transfer. Instead, it would ease the red network to perform some retransmissions when such thing happens. Taking this in account, the parameters for the red network can be distinguished as follows:

Controllable parameters

The red part of the diode is active process. It initialises the TCP stream and is responsible for the data flow. It should maintain a reliable and fast connection, while featuring the TCP characteristics. The process has the task of transferring the data from the diode to the receiver, hence, it is in control of the following configuration parameters:

- **Window size**
  The red source transfers segments from the buffer of the diode to the red receiver. It maintains flow control by the sliding window mechanism. The window size is managed by the CA algorithm as described below. Commonly, the window size is chosen between two values: the congestion window and the receiving window. The red part of the diode is able to increase the window size to increase the throughput if necessary.

- **Congestion avoidance algorithm**
  Congestion avoidance algorithms should prevent the connection to break down too quickly and drop the throughput to a minimum level. The algorithm does also provide a method to take a fair share of

the network resources for the connection. From Equation (3.6) can be concluded the algorithm has a significant impact on the throughput. The CA algorithm is most dominant for the C value of the throughput determination.

- **Recovery algorithm**
  The recovery algorithm is a part of the CA algorithm, but can be enabled separately. The recovery algorithm determines the b value of Equation (3.6). This represents to what extent the alogrithm is able to recover a connection on time. If packet losses are detected early and recovered efficiently, the throughput should be recovered in a short time and the amount of data in the buffer should not have increased much.

- **TCP options**
  The options available in the TCP header provide the protocol to extend its features. The protocol can be enhanced on its reliability, packet loss detection, RTT estimation, bandwidth estimation, TCP-friendliness, throughput performance, recovery or any combination of those. For example. TCP SACK should improve the packet loss detection and recovery and will improve the value of the b in Equation (3.6).

Dependable parameters
The red part of the diode is responsible for transferring data from the diode buffer to the receiver in the network. Transmitted packets are not guaranteed to receive correctly. Packets can get lost due to network anomalies and the sender has to negotiate the TCP configuration with the receiver. Therefore, the red part of the diode is dependent on the following parameters:

- **Round-trip time red network**
  As can be concluded from Equation (3.6), the RTT is a significant value for the throughput. The sender is able to compensate for variations in the RTT to maintain the steady throughput. Unfortunately, it is difficult to estimate the RTT and this requires complex calculations. The RTT is dependent on network architecture, including distance between the devices. It is almost impossible to predict the exact RTT in advance and furthermore, the RTT will vary over time.

- **Availability of TCP configurations on receiver side**
  During the 3-way handshake of the transmission control protocol, the devices negotiate on which settings and TCP options are going to be used. The red part of the diode could desire advanced settings, but if these are not in compliance with the receiver, these settings are unnecessarily implemented in the design. The following parameters are required to work in compliance:

  - **Window size**
    The sliding window system establishes the data flow for the TCP stream. The size of this window is dependent on two independent windows: the congestion window of the receiver (which is determined by the CA algorithm), and the receiver's window. The receivers window is determined by the receiver and commonly linked to the available buffer space for that connection. The minimum value of these two windows will be chosen as the operational window size. If the receivers window is the smallest one, will that be the limiting factor for the throughput for the red connection.
  - **Segment size**
    The maximum segment size (MSS) is negotiated, and the smallest value will be chosen. When the algorithm of Naglé is applied (which is common since the introduction of default TCP NewReno), the TCP sender will almost only send segments with the size of one MSS. Because the data being send has its origin from another TCP stream (from the black network), these segments are likely to have the size of one MSS from the black network. If the receiver on the red side has a smaller MSS, the segments have to be fragmented at the red sender's side. This will introduce extra overhead and comes at the expense of the efficiency of the connection and will slow down the throughput.
  - **TCP options**
    Today, more than 30 TCP-header options are standardised and registered by the IANA [29]. These options are improvements for specific features of the TCP protocol. Options could enhance reliability, packet loss detection, RTT estimation, bandwidth estimation, TCP-friendliness, throughput performance, recovery or any combination of those. Nevertheless, TCP options can only be used

for a connection if both devices are capable to handle these and agreed upon this during the negotiation. TCP options like TCP SACK could provide promising performance improvements as is explained in Appendix A.

- **Packet loss red network**
  Packets lost in the red network will have the same causes as for the black network as described above. The consequences are almost equal for both networks. Packets will get lost and the throughput will slow down. If this happens at the black network, than the black source should detect the loss and recover the connection. This will result in a drop of the throughput, potentially lower than the red throughput. This will create time for the red network to empty the buffer. If a packet is lost at the red network, the same will happen for the throughput of the red network. It will slow down. However, the black network will keep transmitting data and this will stack up in the buffer, while the red part of the diode is recovering the connection and trying to increase the throughput. If the connection is not recovered quick enough, the buffer will overflow and data will get lost, forever.

3.2. Simulations

In previous section, the evaluation model for the system was discussed. The operations of the network diode could be separated in two networks. The black (untrusted) network will act as the source and will transfer data from the transmitting device to the buffer of the diode. The red (trusted) network has the responsibility to empty the buffer and transfer data from the diode to the authentic receiver. Consequently, because the transfer protocol is TCP, two separate TCP streams are active. The black stream is the supplier of the data and will stack information in the buffer of the diode, while the red stream acts as the consumer and will unload the buffer. This resulted in Equations (3.1) and (3.2) as was discussed in previous section. From Equation (3.2), we concluded to find a method to determine the throughput, because there is no standardised method for this. A formula was proposed, which could be simply implemented in our system. This resulted in Equation (3.6), which contains only variables that are straightforward to be controlled by the parameters of our system, or to what variables it is dependent on from the network. The wide range of varieties available from this model motivated us to analyse this system by simulations rather than solving it by mathematics.

Therefore, a suitable simulation software should be chosen for our system. The simulation model implementation is depicted in Figure 3.3. Each part of the network is partitioned in three processes to simulate. Each part has a sending process, receiving process and a process that simulates the network anomalies. In addition to this, the simulator needs a unidirectional connection and a matching buffer to simulate the diode functionality. Therefore, this simulation software should satisfy the following list of requirements:

- **Implement infinite data source and sink**
  The simulation should be able to analyse the diode performance under realistic network environments. The worst-case is a never-ending data stream, which will utilise maximum resources and requires the maximum available capacity of the network. Therefore, the black source must be able to generate a TCP stream of an infinite amount of data, which is transferred at the maximum available sending rate. Finally, this stream will be received at the end-point of the red network. This process should receive all incoming segments and handle them by sending proper acknowledgements.

- **Implement data-diode**
  The main function of this system is providing a one-way gateway for TCP streams. Since this is a new concept, this functionality should be able to be implemented in the simulation software and must comply to our system's configurations. This includes the buffer we want to analyse. From Equation (3.1), we want to know the necessary size of the buffer to use in our design. The simulation should give some insight on the appropriate size for the buffer.

- **Adjustable network resources**
  Our system will be designed to operate under realistic network environments. Therefore, the simulation software should be able the simulate the behaviour of the network environments. In previous section was explained what parameters are of significant order for each network. The behaviour of the packets inside the network must be able to be manipulated. The following list of parameters should be configurable:

  - Packet loss probability
  - RTT variations
  - Other traffic streams
  - Packet reordering
  - Packet corruptions
  - Network congestion

- **Configure TCP parameters**
  TCP has many parameters, options, algorithms, settings, etc. to be configured as was explained in Section 2.2. The simulation software should allow us to explore the consequences of the set of configurations. Therefore the following parameters should be adjustable:

  - **Header segments** The TCP header contains three different types of information: control signals and status signals and some fields to verify the correctness of the message. The status signals are used for the basic mechanism of TCP, acknowledging packets and notifying the status of the connection. These consists of the flags, the matching sequence acknowledgement numbers, address fields. These simulation software must allow us to control these manually. The control signals consist of data which will change the current flow. The window size and the TCP options. The window size is significant to be in charge of the sliding window system.
  - **Window size** The window size is a significant factor for determining the throughput (Equation (3.6)). The simulation should verify the influence of the window size with regards to the buffer space required (Equation (3.1)). The congestion window is managed by the congestion avoidance algorithm, while the minimum window size will be used. The simulator should be able to set limitation value to the window size for analysis purposes.
  - **TCP options** The simulation software will be used to explore the available configurations. There are more than 30 approved TCP options available [29], which will be discussed to test and find out whether they will improve the data diode's performance. Special attention is to the availability of TCP SACK as is explained in Appendix A. The simulation software should allow us to implement the TCP options.
  - **Algorithms** As was discussed in previous section and concluding from Equation (3.6), the algorithm has a significant factor in the determination for the throughput. Both CA algorithm and recovery algorithm can be adjusted to optimise the performance. The CA algorithms we want to analyse are NewReno and Cubic as was discussed in previous section. The recovery algorithm we want to analyse is fast recovery and TCP FACK (see Appendix A. The simulation software should allow us to configure these settings and algorithms.

The network simulator should satisfy the above list of requirements. Numerous network simulators are on the market, all featuring unique selling points. Network simulators provide the possibility to test network modules under realistic network environments. These include physical and software units. The simulation is software and every piece should be modelled in the simulation. Simulation solutions can be distinguished in level of abstraction, complexity, architecture and configurations. A very common type of network simulator, is a Discrete Event Simulator (DES) [57]. Computer network traffic is in general simulated as a series of discrete events. A DES simulates a number of events that occur at discrete time steps. Furthermore, network emulators are being used for simulation purposes. An emulator imitates the behaviour of pieces of hardware devices operating in a network. The emulator itself is a piece of software or dedicated hardware to simulate the behaviour of the device it imitates. The input of the emulator should have the same output as if it was on the real device. From the long list of available network simulators, three different network simulators are picked, based on features, costs and academic recommendations by Dr.ir. F.A. Kuipers from the TU Delft. In the section below the following simulation software methods are considered: MiniNet, OmNet++ and NS3.

3.2.1. MiniNet

MiniNet is an emulator of network and network devices. It provides a platform to deploy large networks on limited resources of a single computing machine or it can run on a Virtual Machine. "Mininet provides convenience and realism at a very low cost" [36]. It allows creating topologies of large network architectures, including a numerous amount of nodes, and perform tests on them very easily. MiniNet is an open source software

project that emulates OpenFlow devices and software defined network controllers. OpenFlow is a standard communication protocol that allows network devices to control the path of network packets across a network of switches or routers. MiniNet has an user-friendly interface by providing simple command line tools and an API. Consequently, does MiniNet allow the user to create custom topologies using Python. Python has a MiniNet module which defines topology classes and functions to add network devices. The simulation model can create a virtual network controlled by the main controller. After the network is deployed, the user can access all network devices to execute processes and applications. We chose to experiment with the Scapy and . Scapy is packet manipulation and sniffing tool written in python. Scapy is provide with an interface with the raw sockets and allows the application to send and capture packets. The TCP/IP suite is available in Scapy and it is capable to create TCP segments, including all header fields. It provides functions to capture packets and structure them as TCP segments and vice versa can it build packets with provided information blocks.

3.2.2. OMNeT++

OMNeT++ provides a simulation tool for multiple purposes. It can be used to model different architectures, protocols, networks, multiprocessors and also hardware validation [68]. The basic of OMNeT++ is a very generic architecture for designing network components and devices. The models are created from small universal modules called a *simple module*. Gates can be attached to a simple module, which allow them to connect to other gates from other modules. Simple modules can pass messages over a route which is connected to gates. Simple modules can also send timed messages to itself to schedule events. Simple modules can be combined to create a *compound module*. This method can be used to create complex structures. The TCP/IP suite can be created by designing every layer from OSI-model and combining them to a compound module. The INET framework, included in OMNeT++ version 5 or higher, has most of the internet modules already designed. The structure of simple modules are written in a NED file, while the behaviour of the modules are written in C++. This abstraction level allows the user to use all features of object oriented programming together with complex designs with many nested modules. Each module, written in a NED file, may contain any of the following sections.

- **Types**
  Types define local variables used within the NED file. This could be settings for connections in between sub-modules.
- **Parameters**
  Parameters are variable that belong to the module. They can be used to set attributes and are accessible outside the module. This could specify attributes such as protocol, packet length, window size, etc.
- **Gates**
  Gates are connection ports to interact with other modules. Connections are assigned to gates specified by the modules.
- **Submodules**
  Combining multiple simple modules will create a compound module. The simple modules inside the compound module are referred as submodules.
- **Connections**
  Connection specify the characteristics of the connection channels between gates. A connection has attributes such as direction, bandwidth, type, delay, etc.

The topology and data used for the simulation is specified in a configuration file. Multiple configurations could be scripted here. The file is a list of key-value pairs. The *keys* are the parameters of the module or settings for the simulation, like number of runs and run-time. This is where settings are configured like what type of protocol is used or how many data will be send. Data collection is managed by setting recordings messages. These recordings are stored as *vector* or *scalar*, based on the signal it records. Signals are emitted by modules and registered in the C++ files. Besides using signals as statistical records, they can be used to interact between appropriate modules. The structure of the signal can be chosen by the developer. This allows the user to track messages in the format of TCP segment for example, by the use of pointers to the TCP segment class. In this way, messages can be structured to have usable fields for the protocols and modules can react to them if properly coded in the C++ file of the module. OMNeT++ provides a tool to analyse the recordings. It features options to show graphs or statistics to quick get the count or maximum value of a

signal. The recordings can be extracted to CSV-files for further analysis by third-party software. The INET framework provide features to extract the data in the form of PCAP-files. The OMNeT++ package is provided with an IDE, which allows the user to analyse the recorded statistics. This is an user interface that makes it possible to show the modules in a graphical presentation, providing a clear overview of sub-modules and connections. The IDE can switch easily between source code and graphical design view.

3.2.3. NS-3

Network Simulator 3 is a new simulator based on the NS-2 simulator, developed in 2005 [68]. The goal of NS-3 was to improve realistic modelling. To achieve a good abstraction level, C++ has been chosen as the programming language. Most protocol stack implementations are implemented in C, which makes it easier to implement these in the NS-3 simulator [57]. NS-3 is an open-source platform, which allows a lot of different people to contribute to the NS-3 library. To create real-world simulations, instances of network models are defined in NS3:

- **Node**
  A node represents an end-point in the network. A network consists of multiple nodes and communication channels will connect the nodes. A node does not specify anything about functionality or hardware.
- **Device**
  A device can be *installed* on a node and represents the physical device. This could be a switch, router, hub, etc.
- **Communication channels**
  The communication channels connect network devices installed on the nodes. These channels represent the medium on which the data is going to be transferred. This could represent a physical cable or a wireless communication.
- **Communication protocols**
  A communication protocol has to be configured on a communication channel. The protocols can be selected from the internet protocols and thus represents protocols like Ethernet, TCP, UDP, etc.
- **Network packets**
  Network packets represent the packets transferring from device to device over a communication channel. The format of the packets is defined by the selected devices and communication channels and protocols.
- **Applications**
  An application is *installed* on a network device. The application is programmed to send or receive network. Traffic is generated by the applications.

To assist the user in creating and analysing models, NS-3 includes practical solutions: attributes, helper objects and trace sources.

- All models are provided with *attributes*, which the user can change to values required for their simulation. The behaviour and configurations of class instances can be set by provide functions.
- To save the user some time and complexities, *helper objects* are included in NS-3. Common tasks are assisted by this. A typical operations to set up a network can be handled by the helper objects. Instead of creating many nodes with the same device and scripting all the connections, the helper object can do this for you.
- The goal of any network simulation is to generate a data flow that can be analysed. *Trace sources* can log (meta-) data of the network packets. The trace sources have to be assigned to other pieces of the code and will provide traffic generation. The data flow is consumed by trace sinks. Both source and sink will log information which can be analysed later on. The trace sinks can be presented in realistic formats, like PCAP for example, if the right package is installed.

The NS-3 is a discrete event simulator (DES) and provides C++ instances to create a simulation model. The classes as defined above has attributes to be configured to your own requirements. Helper objects let's you create simulation models more quick and easier. The analysis of the data flow can be extracted from generated standard formats like PCAP.

	DES/Emulator	High Speed	TCP features	Modular	Friendly UI	Rank
MiniNet	Emulator	–	0	0	+	3
OMNeT++	DES	++	+	+	++	1
NS-3	DES	+	++	0	–	2

Table 3.1: Trade-off overview of simulation software

3.2.4. Software selection

In previous section, the basic concept of three simulation methods was explained. As was described in Section 3.2, two main solutions for network simulations are commonly used: network emulators or Discrete Event Simulators.

First, we explored the possibilities of using MiniNet in combination with Scapy. We created a topology of the network, as is depicted in Figure 3.3. This script can be found in Appendix B. Four hosts are configured, two for each network. Ethernet links are attached to connect the hosts. This does not represent an actual diode. Hence, since we can control the traffic, there will only be a one-directional flow simulated. The black network was simulated by a source and sink which will copy the data over the "diode". The basics of TCP were handled by the python script. Scapy does not provide automatic processing of TCP segments, therefore all TCP mechanism were implemented in the script. At the red side, when a packets was received it will set up a TCP stream to the last host and will initiate the transfer. Small segments could be sent by using the netcat command of linux. However, the process speed was limited by the hardware resources of the running machine. No high sending rates could be achieved.

Second, OMNeT++ was researched. The INET framework provides all instruments to create our system. A *standard host* is a module in the INET framework, representing a network device. This standard consists of modules stacked as is in the OSI-model. Every layer of abstraction adds an extension to the header of the packet. Most functionality of TCP can be configured. A correct operating TCP stream could be achieved from the black source to the red receiver. The output of the records could be extracted to an CSV file and easily be processed and analysed.

Third and last, NS-3 was tested. NS-3 simulation consist of a large C++ project. To compile the source code of your project and make use of the *make* tool, the build system Waf is used on NS-3. In NS-3, the model is created by calling the helper objects. Our system consist of multiple sockets, which needed to be configured with the TCP protocol. The complex nested C++ modules were difficult to modify to a operating data diode in a timely fashion. A bulk send application was used to generate traffic and most TCP protocols could be configured. The results were stored in a large PCAP file, which was heavy to process for our machines.

All three simulation tools were researched and tested as discussed above. The emulator was too dependent on the resources of the computing machine to achieve consequent and high performance. NS-3 provides a broad range of possibilities to create models and modules. However, the interface is complicated and requires valuable time to learn. OMNeT++ has a friendly graphical interface and a well documentation on the framework. It has less integrated TCP features built-in, but lets the user create modules for this at more ease. Therefore, OMNeT++ is the favourite simulation software and will be used for our system, which we will elaborate in Chapter 4.

3.3. Conclusion

In Chapter 3, a suitable model was proposed for system evaluation purposes. The conversion of TCP to one-way traffic (and reversed) will be processed on the same device as the diode is implemented. Together with the basic operations of TCP described in Chapter 2, and taking in account goal 1 (Section 1.3), a simplified model is presented in Section 3.1. This model consists of two separate TCP streams on both sides of the diode. The unsafe network is referred to as the black network and the trusted network is referred to as the red network. Because no feedback is possible, the two networks would operate separated. The only shared resource is the buffer. This buffer is filled with segments from the black network and emptied by the red network. To amount of data inside the buffer is almost equal to the window size of the red network and difference in throughput of both networks as stated in Section 3.1.1. Therefore, the throughput from the black network is not allowed to exceed the red network. To determine the throughput for a TCP stream, a formula is proposed in Section 3.1.2, which consist of four parameters. In Sections 3.1.3 and 3.1.4 is discussed which parameters can be controlled on the diode for each side and to which parameters it depends on from the network.

To evaluate those set of parameters, a network simulation will be used. To select the most suitable simulation software, several network simulation solutions were discussed in Section 3.2. Commonly, two different types of simulators are considered: A Discrete Event Simulator (DES) or a network emulator. We tested three potential simulators: 1) Mininet 2) NS-3 3) OMNeT++. The emulator provided by MiniNet, lacked the capacity to simulate high-speed transfers or process packets in a short time, because an emulator simulates real-time network traffic and requires sufficient resources to do the calculations, which our computers lacked. Both NS-3 and OMNeT++ are DESs and have implemented options to simulate TCP streams. NS-3 models networks by creating a network of nodes and corresponding connections. Network devices can be installed on those nodes to create a real network. Applications can be assigned to the network devices, which includes TCP sockets. The project needs to be compiled by an build system called *Waf*. OMNeT++ uses simpler block structures to design a model. It makes use of simple modules, which could be used for sending any kind of message. It is provided with an user-friendly GUI, which makes designing more comfortable for a beginner. OMNeT++ version 5 or higher is provided with the INET framework, which consist of common network components, including TCP. In addition to this, OMNeT++ IDE has an integrated analysis tool for the output statistics. Therefore, Omnet++ is selected as best simulation tool.

4

Simulation model

In Chapter 3, a simplified model is developed for system evaluation. From this model, Equation (3.6) could be formulated. In Sections 3.1.3 and 3.1.4 the set of parameters for each side of the network are selected based on dependency and controllability. To create a testing environment for these parameters, a suitable simulation software is selected in Section 3.2. OMNeT++ is selected because of its ability to develop specific modules in an user-friendly interface and the existing TCP features available in the INET framework (Section 3.2.4).

First, the model of the network diode must be implemented in the simulation software. The INET framework from OMNeT++ will be used for this as was discussed in Section 4.1. The modules of the INET library has a similar structure as the OSI-model, which can be modified to the simplified model described in Section 3.1. The new module features the functionalities of the network diode and is able to configure the parameters as defined in Sections 3.1.3 and 3.1.4. The testing order of the simulation parameters will be selected on priority and dependency and is discussed in Section 4.2. Each parameter is discussed in high detail and is reflected on goal 1 of this thesis (Section 1.3).

4.1. Overview

To examine the behaviour of the TCP streams through the diode, at first the diode must be implemented in the OMNeT++ tool. The INET-framework is provided with the *standard host*, representing a general network device. The standard host can be configured to send and/or receive messages, using multiple protocols over multiple media. For our system, we are only interested in the TCP/IP suite transferred over an Ethernet cable. OMNeT++ uses simple modules to create and manage messages. Combining multiple simple modules creates module compound. This is exactly how the standard host is modelled. In Figure 4.1 a graphical overview is depicted of the standard host as defined by OMNeT++.

The OSI-model (Figure 2.4) and Figure 4.1 show many similarities. At the bottom of Figure 4.1, the physical and datalink layer is represented. Here are sub-modules for Ethernet, WLAN and point-to-point devices implemented. These sub-modules allows standard hosts to be connected to each other. For our system, only the Ethernet module will be used. This module receives packets structured as Ethernet segments. It will extract the MAC-address and the protocol to pass this to the next layer. This process also works the other way. The Ethernet module can receive a message from the upper layer and will parse this into an Ethernet packet to send. The small purple bars in Figure 4.1 represent a message dispatching interface. They function as the gateway between two layers. The lowest bar dispatches messages from the data-link module to the network link modules. It checks for protocols, which should be registered by the developer, to pass the message to the correct sub-module. The second layer represents the network layer. In the standard host ipv4 and ipv6 are implemented. This is where the MAC-addresses are linked to an IP-address. The interfaces are being stored in the interface table. The network-layer devices will extract the transport protocol and pass the messages to the upper layer. Three protocols are in the standard host of the INET-framework: UDP, TCP an SCTP. For our system, we will only look at the TCP module. The TCP module processes TCP segments according to RFC 793. The module implements the following features of TCP:

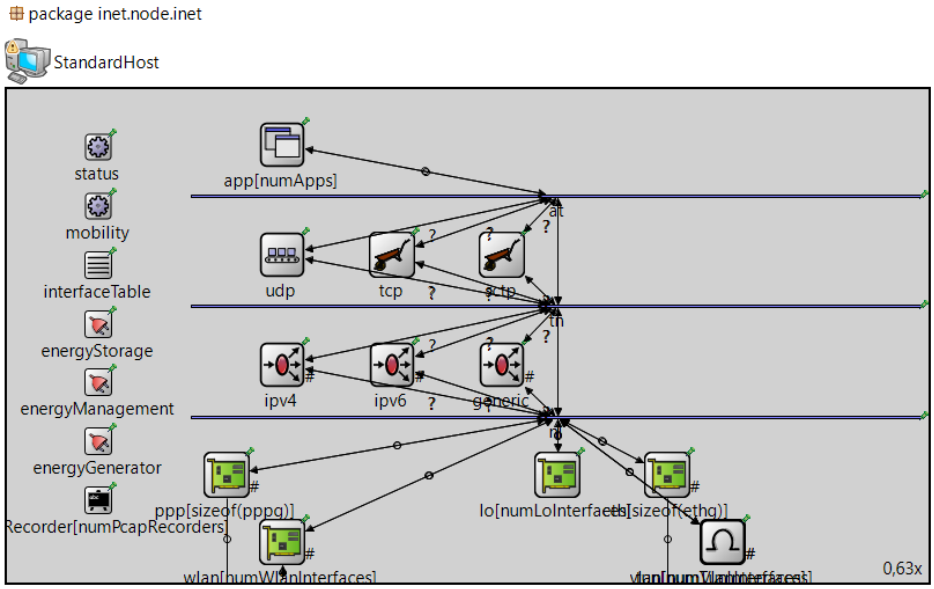


Figure 4.1: Graphical representation of the standard host defined by OMNeT++

- **RFC 793** Transmission Control Protocol
- **RFC 896** Congestion Control in IP/TCP Internetworks
- **RFC 1122** Requirements for Internet Hosts – Communication Layers
- **RFC 1323** TCP Extensions for High Performance
- **RFC 2018** TCP Selective Acknowledgment Options
- **RFC 2581** TCP Congestion Control
- **RFC 2883** An Extension to the Selective Acknowledgement (SACK) Option for TCP
- **RFC 3042** Enhancing TCP's Loss Recovery Using Limited Transmit
- **RFC 3390** Increasing TCP's Initial Window
- **RFC 3517** A Conservative Selective Acknowledgment (SACK)-based Loss Recovery Algorithm for TCP
- **RFC 3782** The NewReno Modification to TCP's Fast Recovery Algorithm

The TCP module is able to pass payload to the upper-layer for processing by an application or it can receive data from an application to be transferred. Multiple applications and protocols can be set active on the same host. The unmentioned sub-modules at the left are not of significance in our system. Except for the recorder, this module is able to create PCAP-files of the simulation runs, which can be used for analysis. All behaviour of the modules are written in C++. The topology of the network and all parameters are defined in the OMNeT++ project files (.NED or the configuration file). A common topology of a network featuring a TCP stream, could exist of two standard hosts. One acting as a client, sending requests to the other host, acting as a server. The first host has a TCP-client application configured. The set of parameters and the settings used for the simulation are scripted in the configuration file (.ini files).

4.1.1. TCP diode design

The module of the standard host allows us to create a model such as described in Figure 3.3. Three standard hosts will be used for this. The first host will be the black source, the second host serves as the data diode and the last host will be the red receiver. To create the black source and the red receiver, the existing standard host can be used without any modifications. The lower sub-modules (Ethernet, IPv4, TCP) provide the interface to let an application send unlimited data. The black source should be able to simulate the dependable

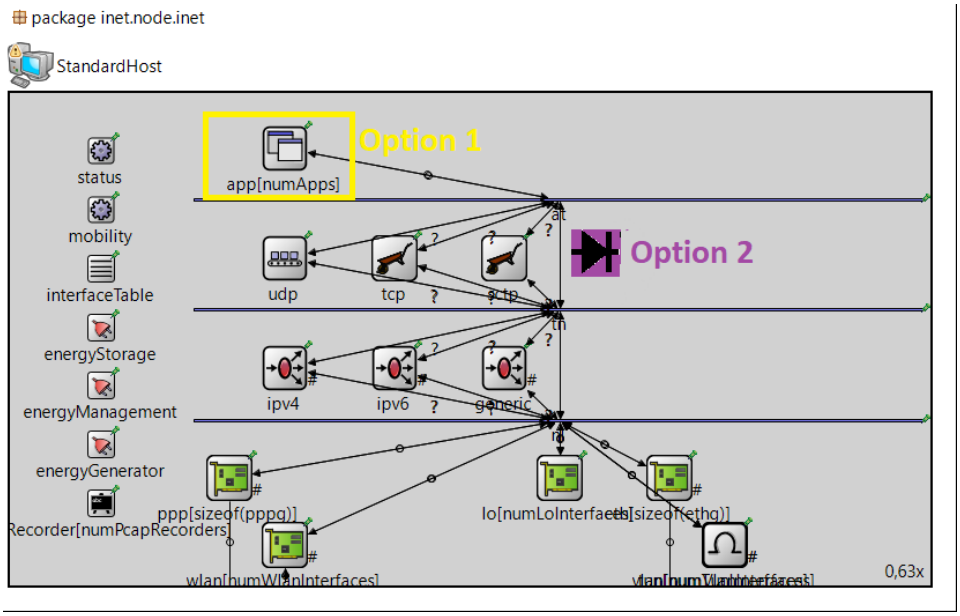


Figure 4.2: Available options to implement the data diode

parameters of the black network as was discussed in Section 3.1.3. The **sending rate** and the amount of data to be send, can be configured by the *TCPSessionApp* of the INET-framework. The TCPSessionApp can create a single-connection TCP application. It will establish a connection and will send a pre-defined number of bytes. The **congestion avoidance algorithm**, as well as the **TCP options**, can be set in the configuration file of the simulation. In addition to this, the **packet loss** can be configured by setting the *packet error rate* in the connection between the hosts by a *channel* module. A channel has configurable data-rate, delay and packet error rate. The red receiver should be able to simulate the dependable parameters of the red network as was discussed in Section 3.1.4. The receiver host will be simulated with the *TCPSinkApp*. This application accepts any number of incoming TCP connections and will respond to them properly. The **TCP configurations** will be set in the configuration file of the simulation. The **RTT** and the **packet error rate** can be configured by the channel, which connects the diode host to the receiver host. The host featuring the diode, should simulate the controlling parameters of both the black and the red network (see Sections 5.2.1 and 5.2.2), and should simulate the management of the buffer. The segments received from the black source should be stored in a buffer and the other process in the diode host should empty the buffer by transmitting the data to the red receiver. We propose two suitable modifications to the standard host module as implementation method for the diode.

Option 1: Create *diode application* at upper level

The first option is to create a new TCP application to run. The application can be configured to run on the standard host. TCP applications are connected to a TCP socket from the lower level. The TCP port number and IP address of the host can be configured. If source and destination information match the settings of the application on the host, the messages will be passed to the application. A TCP application is also able to start a connection from a connected socket. The payload and destination information can be set in the parameters of the application. From the application level it is not possible to adjust parameters for the TCP sockets. The parameters for the TCP sockets can be set in the configuration file of the simulation. However, the application can only interact with TCP sockets with regards to the payload of the messages. It can define to what socket it should listen or to send to, but the application can not interact what happens to the TCP segment when it arrives. All the TCP mechanisms are handled by the TCP module. These mechanism process the segments, send the acknowledgements, manage the state of the connection, etc.

Advantages

- Easy to implement
- Most common TCP features available

Disadvantages

- Modifications to TCP features not possible

Option 2: Create new *diode module* in OSI-layer 4

The other option is to create a complete new diode module on the transport level. The modules on this level receive messages from the lower layer if the packets are registered as a TCP message. Creating a complete new module on this abstraction level, allows us to have complete control of the TCP parameters. All TCP mechanisms can be designed in this module. However, most of the TCP segment handling that our system requires, is already implemented in the TCP module of the INET-framework. This mechanisms includes processing TCP segments and create proper responses for this. When a segment arrives in a TCP socket, the packet will be verified, the ACK number is checked, the TCB is updated, the window size is recalculated, the congestion window is updated, the flags are checked (except URG and PSH), the state of the connection is updated (Figure 2.6). Any modification for our system, e.g. delayed acknowledgements or artificial packet loss, can be implemented by using this method. However, the current implementation of the TCP module is complex network of nested sub-modules and call-back functions. While the segments are processed, funcions are called from the socket they are assigned to. Implementing a new socket to the process (to simulate the one-way gateway), requires a numerous amount of additive lines of code. Therefore, it will take a large amount of time to append new features in the existing solution. Creating a new protocol next to the existing TCP module does not compile or run. The segments send by the black source will get the TCP-tag by the layer-3 modules (IPv4). The message dispatcher will send these to modules which are registered on the TCP-tag. Nevertheless, only one module can be registered.

Advantages

- Possibility to implement new TCP features

Disadvantages

- Difficult and time-consuming to implement

New Diode Application Module

Our **goal of the simulation** is to analyse the effects of the TCP streams on the required buffer space in the diode. Both options are a suitable solution for this. Since the TCP module of the INET-framework is provided with most common TCP features, there is no need for extra features. Creating an application module to simulate the diode application is not an exact implementation of a hardware diode. Nonetheless, the simulation is already software simulating hardware, and the functionality will be very similar. Therefore, we will choose for **option 1** and create a new **diode module application**.

The complete simulation model is depicted in Figure 4.3 and the source code is presented in Appendix C. The model consist of three standard hosts and two Ethernet connections. This, as mentioned before, does not represent a real data diode, since data can still flow in both directions. However, the simulation will only transmit data from the black source to the diode, and the diode will only copy these messages and transmit them to the red receiver. This is sufficient to analyse the behaviour of the system. The TCP diode application binds two sockets for its purpose. One socket is the receiving socket (black network), and the other will be the sending socket (red network). Every message received from the black network socket will be sent to the red network socket. The TCP socket is operated by the TCP module. The TCP module will queue this message and processes the transmission (and even retransmission) of this segment. The parameters of the TCP sockets can be set in the configuration file of the simulation. The list of configurable parameters in the TCP module almost match the list of desired parameters as was discussed in Section 3.2. Only the header fields are not complete accessible via this method. The header segments are created by the TCP module, while some of the options can be configured. A small set of TCP options and congestion avoidance algorithms can be established for the TCP sockets. Unfortunately, the TCP socket parameters are set for the TCP module of each host. The standard host implementation of OMNeT++ allows only one TCP module to be integrated in a host. However, this does not cause any issues to our system. To use TCP options, both the sender and the receiver should agree to the use of this. Therefore, the TCP options can be configured at the black source and red receiver. The advertised window size is active for the receiving socket. Therefore, configured window size will set the limit for the receiving side of the diode (the black TCP socket).

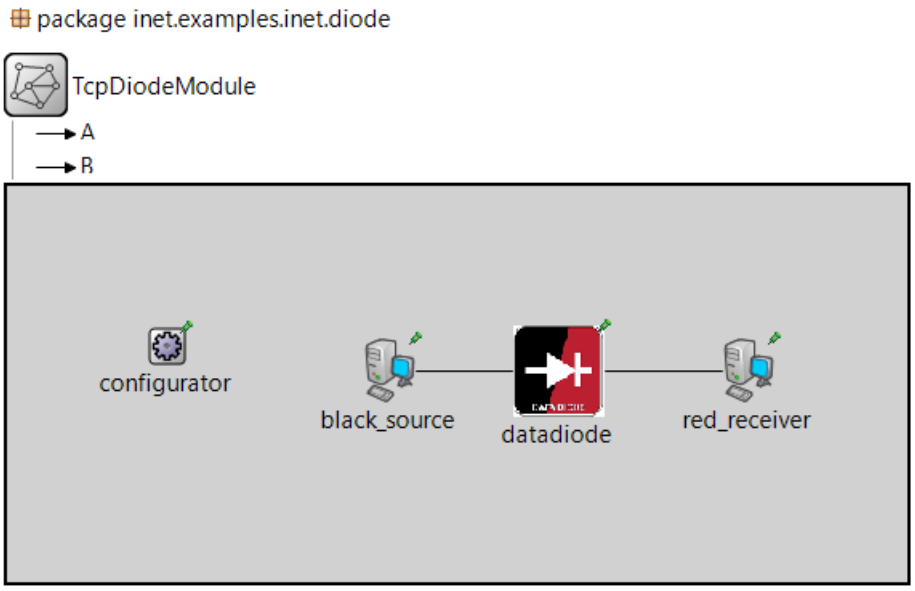


Figure 4.3: Graphical view of TcpDiodeModule.ned

4.2. Parameter study

The goal of the simulation is to analyse the relationship between the network parameters and the required buffer size. Our model of the data diode application does not have a buffer implementation. Instead, the received segments are immediately passed to the red socket. The TCP sockets have an "unlimited" amount of buffer space to manage the transmission of the data. Therefore, we could use the standard recordings of the sockets to calculate the data inside the buffer at a specific time. When the data is received at the black socket of the diode, an acknowledgement will be sent to the black source. This will change the RCV.NXT (Figure 2.8) value of the TCB of the black socket. This segment will be passed to the buffer. The red socket tries to empty to the buffer by transmitting the segment to the red receiver. The segment must be kept in the buffer until the red socket receives an acknowledgement that the segment is arrived correctly. This will update the SND.UNA of the TCB of the red socket. The RCV.NXT and the SND.UNA pointers can be extracted from the simulation recordings. when the initial value for the sequence numbers (ISN) is set to 1, these values does represent the actual number of bytes. The difference in between these two pointers indicates the amount of data which should be in the buffer. An example of this is depicted in Figure 4.4. The amount of data in the buffer will measured over time. The maximum value should give us an indication to what minimum buffer space is required.

Our final design consists of five configurable units. An overview with the parameters for each unit are depicted in Figure 4.5. Only the parameters we are going to use, are shown. Two of the TCP module parameters are being set equal for all units:

- The Maximum Segment Size (*MSS*) will be set equal for units. This value determines the maximum size a segment can have for a TCP stream (see Section 2.2). If the MSS of the black TCP stream is larger than the MSS of the red stream, the segment has to be fragmented and separated over multiple segments. This introduce extra complexity and could be an extra reduction of the red throughput, because this introduces extra overhead. In the worst case, the same payload now does need two headers. The value of the MSS is set to **536** bytes. This is the default value, which is based on the maximum transition unit of the Internet Protocol, which is 576 bytes. Subtracting the length of the IP header and the TCP header (both 20 bytes), leaves 536 bytes for the segment.

- The *WindowScaleSupport* is enabled for all devices. The Standard window size has a maximum of 65.535 bytes, because this is indicated by the 16-bit value in the segment header. This is a small window for achieving high data rates. To achieve a data rate of 1 Gbps with a window size of 65.535, the round-trip time should roughly be 500μs. Therefore, setting this boolean value to *true*, allows the TCP module to enlarge the window. The maximum window size will then be 1.073.725.440 bytes.

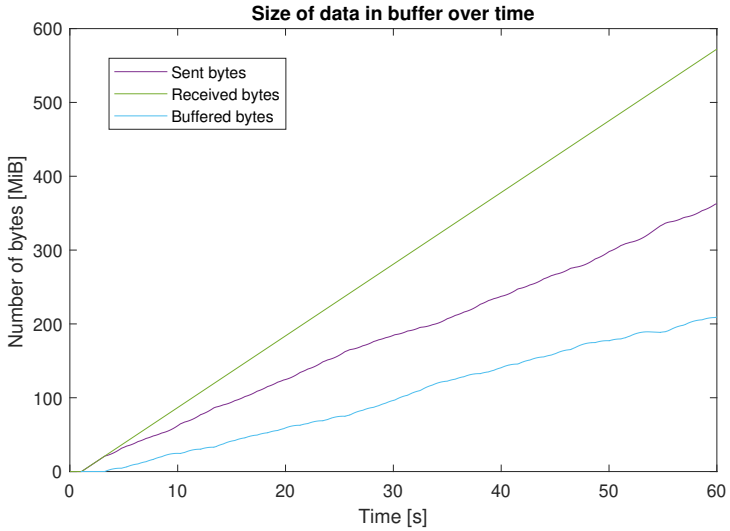


Figure 4.4: An example of the buffered data over time. The green line represents the data segments received in the black socket. The purple line represents the sent and acknowledged data segments at the red network. The blue line is the difference and represents the amount of data in the buffer

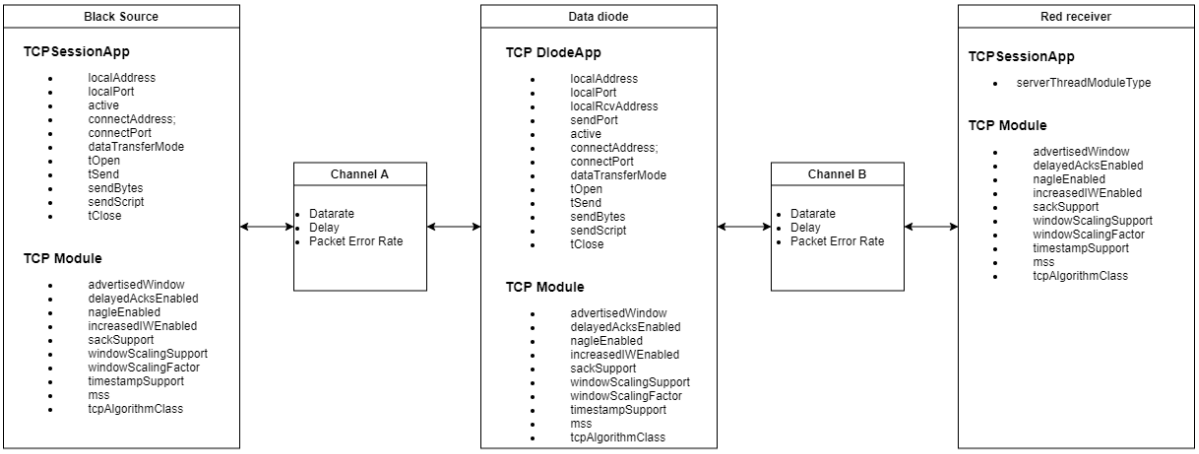


Figure 4.5: Overview of configurable parameters in our design

The other parameters will be set for each simulation run differently. We want to explore the significance and the influence of the parameters to the amount of buffered data. We use Equations (3.1), (3.2) and (3.6) as the backbone for the test-setups. In Section 3.1.1 we formulated Equation (3.2). First, we are going to research this relationship by simulating different throughput at the black and the red network. The black network will be limited to a particular percentage of the red throughput. Secondly, we war going to research the TCP option SACK, which should improve the TCP stream as well as it should reduce buffer space as is discussed in Appendix A. Third, we are looking at the effects of a congestion avoidance algorithm for our system. Furthermore, we will research if our formulated equation about the the buffer size (Equation (3.1)), can be validated and will analyse the relationship between window sizes of the black and red TCP stream.

4.2.1. Ratio of throughput

From Equation (3.2), we want to investigate the relationship between the black throughput and the red throughput. We are testing the effect of the black throughput as a percentage of the red throughput. For example, what amount of data is buffered when the throughput of the black network is 80% of the red network? This will be done for different percentages with steps of 5%. To change the throughput we set the maximum throughput of the red network at a fixed rate of about 80 Mbps. The maximum window size is fixed at 100 KiB (i.e. 102400 bytes). To reduce the throughput of the black network, we will increase the delay of the channel, which will increase the RTT and thus decrease the throughput (see Equation (3.6). The configuration parameters are presented in Table 4.1.

Black source		**Channel A**		**Data diode**		**Channel B**		**Red Receiver**	
Window	102400	Datarate	1 Gbps	Window	102400	Datarate	1 Gbps	Window	102400
SACK	false	Delay	10 to 20 ms	SACK	false	Delay	10 ms	SACK	false
timestamp	false	PER	0 or 0.0001	timestamp	false	PER	0 or 0.0001	timestamp	false
Algorithm	NewReno			Algorithm	NewReno			Algorithm	NewReno

Table 4.1: Configuration of the parameters for the ratio test

In the situation no packets are getting lost, we expect the amount of buffered data to be equal to the window size of the red network (i.e, 100 KiB). The ratio difference in throughput will be tested by introducing packet losses. Therefore, the packet error rate of the channels will be set to induce packet losses to the system. Therefore, a small error rate (0.01%) is tested to check at what ratio the red network is still able to operate without a buffer overflow.

4.2.2. TCP SACK

The TCP option SACK is able to improve the TCP's recovery as is explained in Appendix A. The additional information provided by the TCP SACK option (which consists of maximum 40 bytes), could feature three improvements in reducing buffer space: 1) Specific reporting on correctly received segments, even if this is *not cumulative* 2) Reduce risk on congestion, which allows the sender to have a larger congestion window 3) Reduce recovery time, allowing the throughput being restored more quickly. The simulation will only enable SACK on the red side. Disabling this on the black side, unloads the red network, by having more time to restore. The configuration for this test is presented in Table 4.2.

Black source		Channel A		Data diode		Channel B		Red Receiver	
Window	102400	Datarate	1 Gbps	Window	102400	Datarate	1 Gbps	Window	102400
SACK	false	Delay	10 to 20 ms	SACK	true	Delay	10 ms	SACK	true
timestamp	false	PER	0 or 0.0001	timestamp	false	PER	0.0001	timestamp	false
Algorithm	NewReno			Algorithm	NewReno			Algorithm	NewReno

Table 4.2: Configuration of the parameters for the SACK test

4.2.3. Congestion avoidance algorithm

During testing, we noticed the phenomenon of the sliding window system, dictated by the congestion avoidance algorithm, to cause high peaks during buffering of data. In Figure 4.6, the results are depicted for a test which simulated a black throughput at 50% of the red throughput and packet loss enabled. This figure

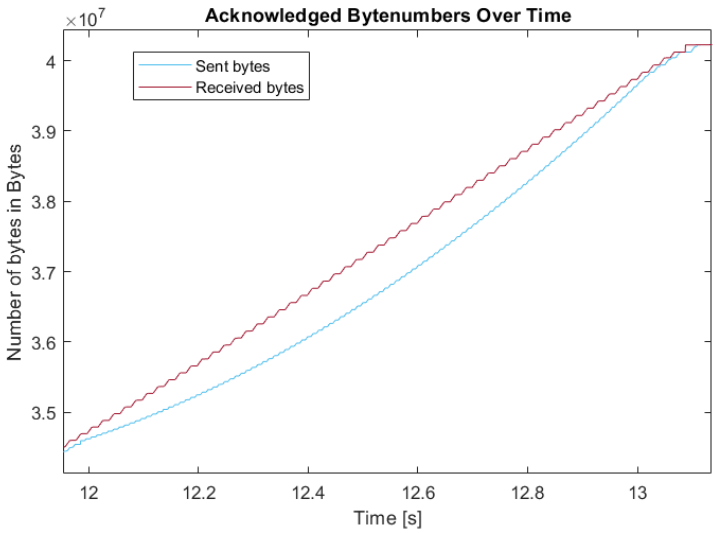


Figure 4.6: Peak buffering induced by the congestion avoidance algorithm

shows the recovery of one packet loss, which costs the red network roughly one second to restore. The bulge between the two lines is caused by the congestion algorithm.

The algorithms reacts to the lost of a packet and reduces the congestion window to prevent more losses. The algorithm is designed to operate in an unknown network, with no knowledge about the connection network and other traffic. The red network is in a trusted domain and in control of the user of the network diode. Therefore, we will test the effect if no congestion control is active. This allows the the window to utilise the maximum window of the receiver, after a lost packet is recovered. This still requires the algorithm to change the congestion window during recovery, to increase recovery time. Therefore, only the C will be affected from Equation (3.6), because the fast recovery is still determining the b value. A test will be performed with the configuration of Table 4.3.

Black source		Channel A		Data diode		Channel B		Red Receiver	
Window	102400	Datarate	1 Gbps	Window	102400	Datarate	1 Gbps	Window	102400
SACK	false	Delay	10 to 20 ms	SACK	false	Delay	10 ms	SACK	false
timestamp	false	PER	0 or 0.0001	timestamp	false	PER	0.0001	timestamp	false
Algorithm	NewReno			Algorithm	NoCA			Algorithm	NoCA
				FastRecovery	Yes				

Table 4.3: Configuration of the parameters for the no congestion avoidance algorithm test

4.2.4. Window size

The effect of the ratio between throughput of the red and the black network is discussed above. For this test we want to research whether the ratio of the windowsize has influence on the buffer when the throughput is equal. In the framework to create a VHDL implementation of the data diode, the window size will be a static value. The windowsizes are set to match the round trip time and always will be able to have a throughput of 100 Mbps. Therefore, a test will be executed with a configuration following Table 4.4

Black source		Channel A		Data diode		Channel B		Red Receiver	
Window	819200	Datarate	1 Gbps	Window	12.5 KiB to 800 KiB	Datarate	1 Gbps	Window	12.5 KiB to 800 KiB
SACK	false	Delay	2.5 to 20 ms	SACK	false	Delay	2.5 to 20 ms	SACK	false
timestamp	false	PER	0.0001	timestamp	false	PER	0.0001	timestamp	false
Algorithm	NewReno			Algorithm	NoCA			Algorithm	NoCA
				FastRecovery	Yes				

Table 4.4: Configuration of the parameters for the no window size comparison test

4.3. Conclusion

The set of parameters in Equation (3.6), which was discussed in Chapter 3, has to be simulated in OMNeT++. Therefore, the model described in Chapter 3 should be implemented in the simulation software. In Chapter 4, the TCP diode module for system simulation was developed. The model can enable all potential features of the network diode as well as the properties of a TCP connection. The network diode module is implemented using the INET framework of Omnet++ as described in Section 4.1. The standard host module, which is a module provided by the INET framework, will run the TCP Diode Application. This application is bound to two TCP sockets: one socket in the black network and one socket in the red network. The application copies all messages from the black socket to the red network. From the statistic records from the sockets, the value of buffered data is acquired. This value will be investigated during the simulations, according to the equations formulated in Chapter 3 (Equations (3.1), (3.2) and (3.6)). The parameters, as discussed in Sections 3.1.3 and 3.1.4, can all be configured in the simulator. First of all, Equation (3.2) has been tested, by researching the relationship between the throughput of the black and the red network with regards to the utilised buffer space. Thereafter, the TCP SACK option will be analysed, after our hypothesis of the improvements for the buffer of the data diode (Appendix A). The influence of a congestion algorithm is questioned to be of a negative impact on the required buffer space. Therefore, tests without a congestion algorithm are performed. Finally, a relationship between the window sizes of the red and black network has been researched to validate Equation (3.1). To **summarise**, the following aspects of the network diode will be investigated:

- From Equation (3.2), we want to know the relationship between the throughput of the black and the red network. What is the impact of the difference in throughput of both networks on the amount of buffered data?
- Does TCP SACK (Appendix A) improve the network diode, by reducing the amount of buffered data?
- Is the congestion avoidance algorithm improving the throughput in a controlled network? Or does congestion window management introduce avoidable buffering?
- What is impact of the difference in window size of both networks on the amount of buffered data?

5

Analysis and results

The most important value to analyse is the behaviour of the throughput for each network side of the diode as is explained in Chapter 3. Especially the relation of the throughput of both networks to the amount of buffered data should be found, because the buffer size should be minimised following the project goals (Section 1.3). Following Equation (3.6) the throughput is determined by several factors. A method to find suitable parameter sets for these factors is described in Chapter 4.

In this chapter the analysis of the results of the tests performed in Chapter 4 will be discussed. Each individual test will be analysed and reflected on the project goals (Section 1.3) . Evaluating over these parameters, this analysis will result in a recommending setting configuration for the network diode. These are separately discussed for each side of the diode in Sections 5.2.1 and 5.2.2.

5.1. Results

The test set-ups of the simulator are discussed in Chapter 4. In this chapter, the results will be analysed and reflected on the project goals (Section 1.3).

5.1.1. Ratio of throughput

The ratio of the throughput evaluates Equation (3.2), which we proposed in Section 3.1.1. This equation states that the throughput of the black network should be less, or at most equal to, the throughput of the red network. If the throughput of the black network exceeds the red network, the amount of buffered data will increase limitless. The window size of is set to 100 KiB and a throughput of 100% is about 80 Mbps. The first test simulates two situations: one situation where there is no data loss in both networks, and one situation where there occurs data loss only at the black network. The maximum amount of buffered data is depicted in Figure 5.1. The results show the amount of data buffered is almost equal to the window size, as we expected according to Equation (3.1). The yellow line indicates a slightly larger required buffer for a situation with packet loss in the black network. This is probably due to the bulks of new segments after the connection is restored at the black network. The situation without losses, decreases for a lower throughput of the red network. The red network can process segments faster than the black network. Therefore it runs in advance and even no full window size is available in the diode's buffer.

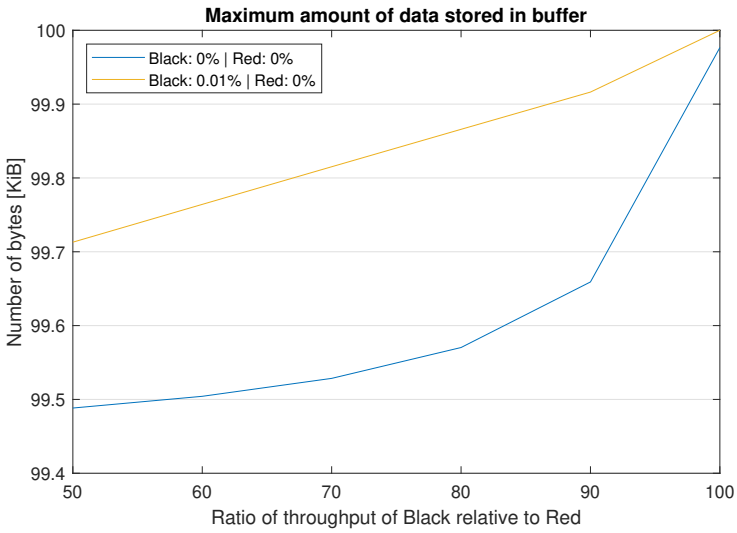


Figure 5.1: The maximum amount of data stored in buffer for different throughput ratio's, when packet error rate was 0% at the red network

Another situation is simulated when there is no packet loss in the black network, but there is only packet losses in the red network at a 0.01%. The amount of buffered data is presented over time in Figure 5.2. Within a one-minute simulation, only tests with 50% and 60% throughput at the black network will have an limited amount of buffered data. The other runs were not able to restore the connection quick enough to empty the peak of buffered data. There, again, is a flat line at about 0.1 MiB, which confirms Equation (3.1).

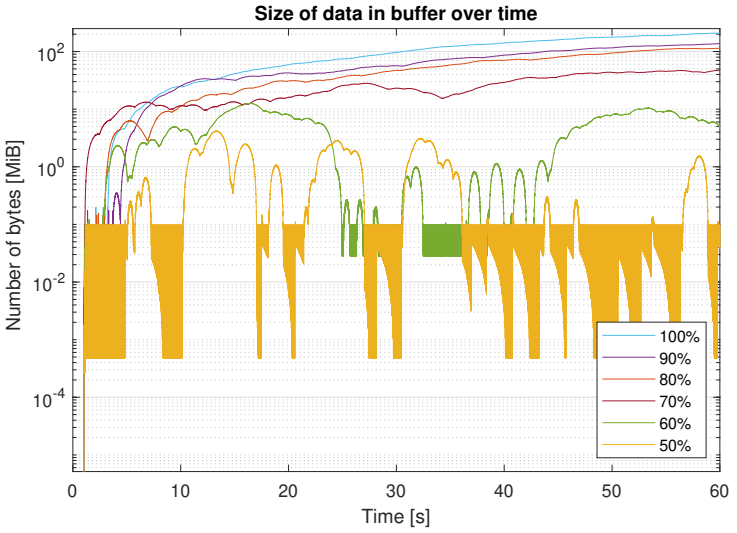


Figure 5.2: Amount of buffered data over time for different ratio throughput of black and red network. Black network has no packet loss, red network has 0.01% packet loss

To simulate a somewhat more realistic scenario, packet losses are induced at both sides of the network. The buffered data over time is depicted in Figure 5.3. For this situation, only the run where the black network was equal to the red network, the buffer would keep increasing. For the runs with a smaller throughput at the black network, the red network is able to restore the connection and reduce the buffered data.

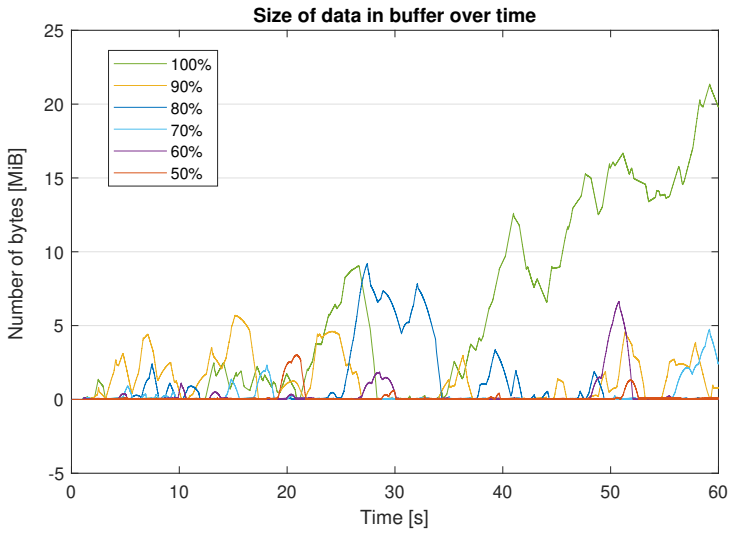


Figure 5.3: Amount of buffered data over time for different ratio throughput of black and red network. Black network and red network has 0.01% packet loss

The maximum values of the buffered data for both situations with packet loss at the red network is depicted in Figure 5.4. From the purple line, looking at the values from the 50% to the 90% ratio, the maximum buffered data is roughly 10 MiB. This is approximately ten times the congestion window.

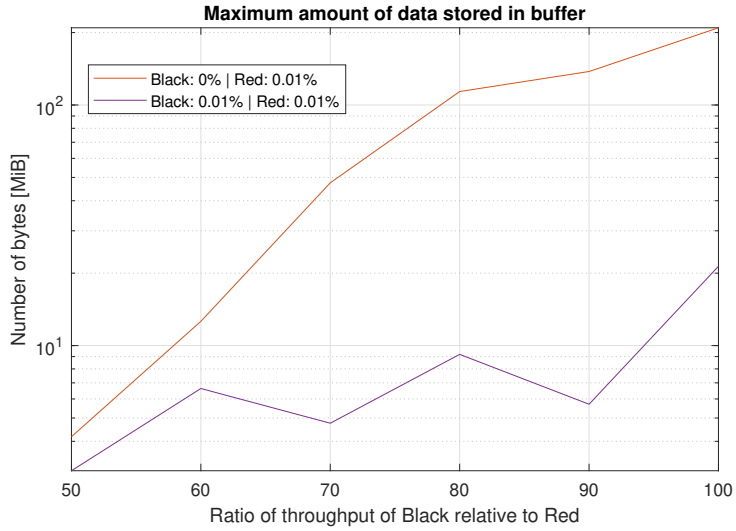


Figure 5.4: The maximum amount of data stored in buffer for different throughput ratio's, when packet error rate was 0.01% at the red network

5.1.2. TCP SACK

The TCP SACK option should reduce the amount of data stored in the buffer, by providing additional information of the correctly received packets. The options is used by defining blocks of correctly received packets. All packets between the edges (indicated by sequence numbers), are correctly received and can be removed from the buffer (Appendix A). Furthermore, this option should reduce recovery time and increase throughput efficiency, which will unload the buffer as well. Two situations are simuluations with the same throughput settings as in the previous test, only now TCP SACK is enabled for the red TCP stream. First situation there is no packet loss in the black network and for the other there simulation there is. The results of these tests are compared to the previous results are depicted in Figures 5.5 and 5.6. The results with SACK being enabled does not show significant improvements with regards to reduction of required buffer space. The amount of buffered data is quite similar.

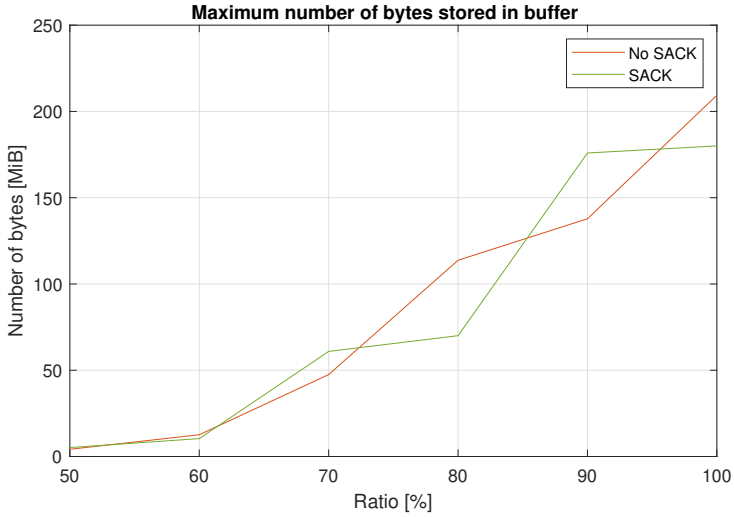


Figure 5.5: Comparison of test with and without sack, without packet loss at the black network

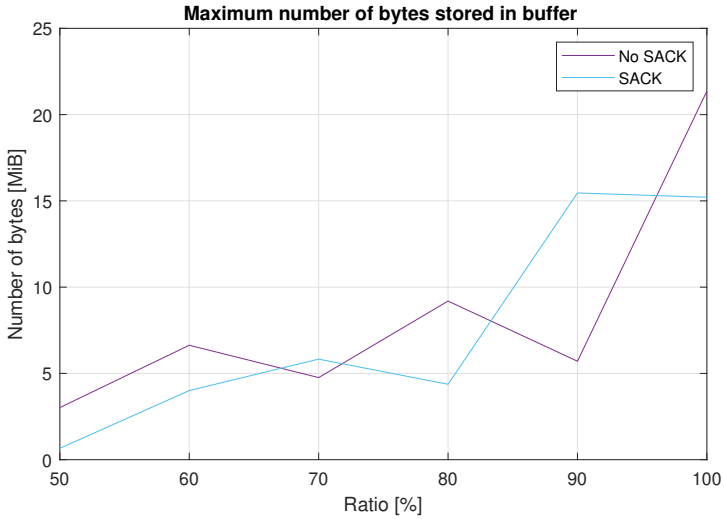


Figure 5.6: Comparison of test with and without sack, with additional packet loss in the black network

A piece of the buffered data over at time at the occurrence of packet loss, is depicted in Figure 5.7. This is from a simulation with a 50% throughput at the black network and packet losses enabled at both sides. At around 11.7s a packet is lost, which is recovered at around 12s. The high arc in the buffer is created due to the congestion window being reduced by the congestion algorithm (Figure 4.6. Therefore, the SACK option does recover the stream in a timely fashion, but does not reduce the required buffer space in the long-term.

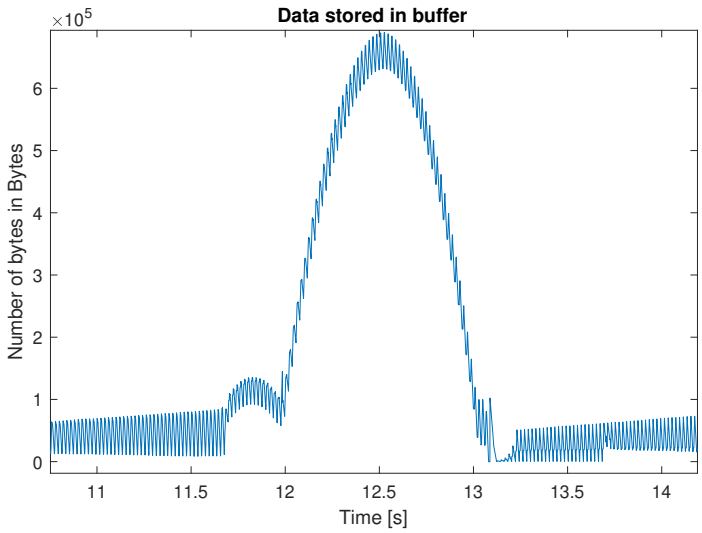


Figure 5.7: The amount of buffered data at a packet loss. Situation of 50% throughput at black side, both networks have packet losses

5.1.3. Congestion avoidance algorithm

The peak in the buffered data, exposed in Figures 4.6 and 5.7, is caused by the congestion avoidance algorithm. For these situations, TCP NewReno. This is the most common used algorithm from the available algorithms implemented by the INET-framework. All the reactive algorithms will reduce the congestion window after a lost segment. The algorithms prevent network congestion and improves the TCP-friendliness of the TCP stream. The sending part of the diode is in the red network. The red network is a trusted domain and in control of the user of the diode. Therefore, the priority of the network diode stream can be given high priority, which allows to disable the congestion algorithm. The congestion algorithm always assumes the packet loss is due to a resource error and therefore reducing the congestion window to sense the network available capacity for that TCP stream. Disabling the congestion algorithm results in disabling the congestion window and only use the advertised window by the receiver. Then still, the congestion window should be adjusted during recovery. Otherwise the sender has to wait one RTT period for each lost packet, which will reduce the efficiency of the throughput drastically and results in a high amount of buffered data (Section 2.2.6).

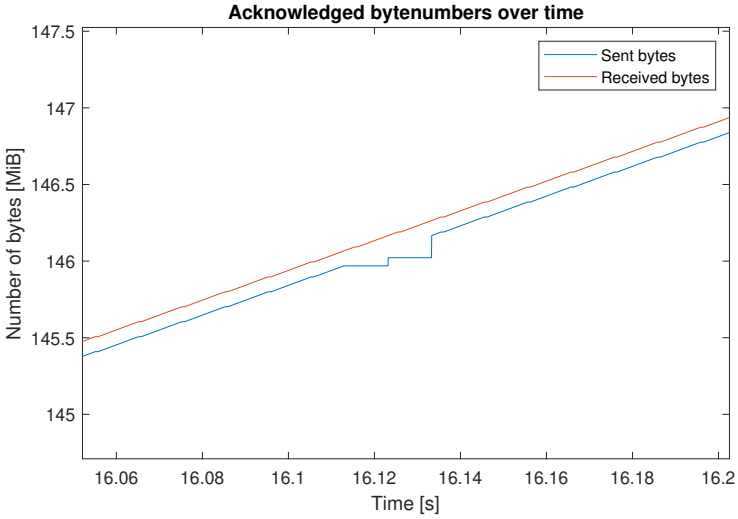


Figure 5.8: The sending window will be restored to the advertised window after the recovery of a lost segment

The recovery of a lost segment, with TCP SACK enabled, is depicted in Figure 5.8. In this example, two packets are lost. The blue line represents the packets correctly transferred to the red receiver, which are cumulatively acknowledged. The line goes flat when a packet is lost. Thanks to the SACK block, multiple packets

can be acknowledged when the the lost packet is recovered. That is the reason for the jump in the blue line. All the blocks acknowledged by the SACK block, can be cumulatively acknowledged when the lost packet is retransmitted. Comparing Figure 5.8 with Figure 4.6, shows there is no peak in the buffer due to the lack of reduction in the congestion window. In addition to this, the connection is recovery more quickly with regards to the available buffer space. The next packet loss will restart the recovery process from the same starting value as the previous packet loss. This is contrary to the situation which is using a CA algorithm. If a consecutive packet loss happens there, the peak of the buffer will even increase more, because the congestion window will be reduced again.

Tests are performed where the CA algorithm is disabled, but the fast recovery algorithm is still active. The congestion window is only adjusted when a packet loss is detected. These test are performed both with SACK disabled and enabled in a situation with packet losses induced in the red network. One test has no packet loss in the black network, the other test has induced packet losses in the black network. The results are depicted in Figures 5.9 and 5.10. The maximum amount of buffered data for both simulation runs, approaches to 0.2 MiB. This is a promising result with a required buffer space of nearly two times the window size. Furthermore, the TCP SACK option does still show no significant improvement of the required buffer space.

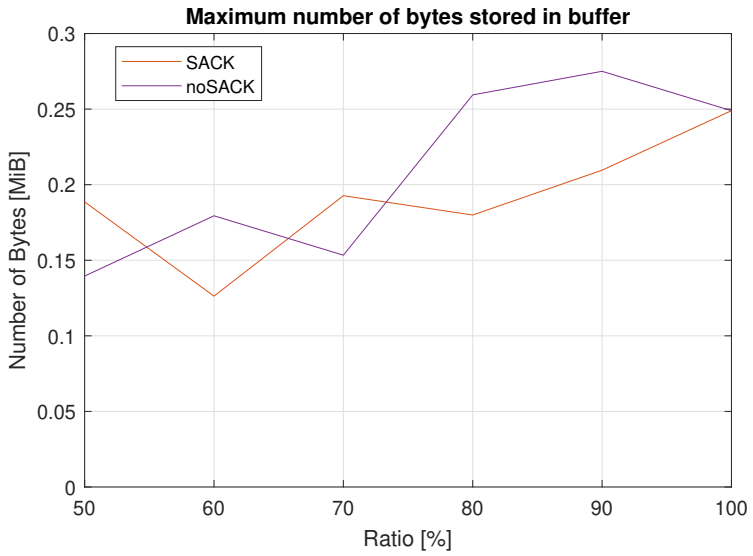


Figure 5.9: Comparison of test with and without sack, without packet loss in the black network and congestion avoidance algorithm disabled

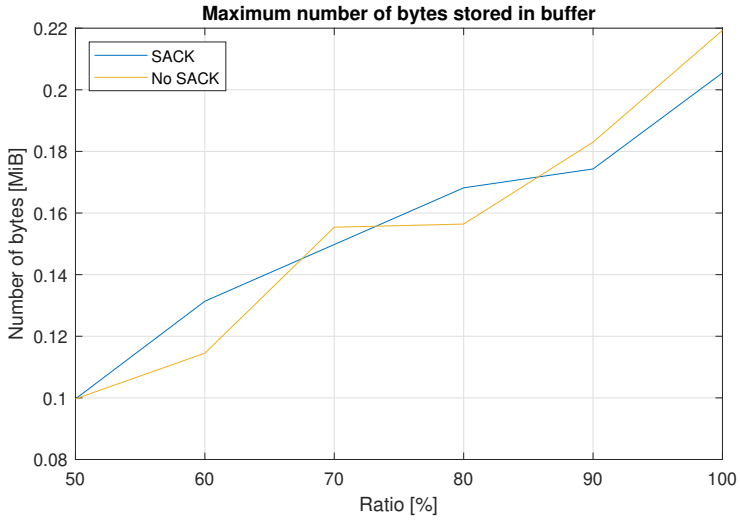


Figure 5.10: Comparison of test with and without sack, with packet loss in the black network and congestion avoidance algorithm disabled

5.1.4. Window size

This simulation will research the influence of the window size when the throughput is not altered. Therefore, a situation is simulated that both networks will have an equal throughput (100 Mbps). Different window sizes will be tested, but at an equal throughput. This is compensated by matching the correct RTT value. There will be no congestion avoidance algorithm configured, as this should be beneficial for the amount of buffered data. The packet loss is enabled on both networks. The results of this are depicted in Figure 5.11. Each different line represent the tests where the window size of the black network is fixed. The x-axis represents the window size of the red network. The amount of buffered data is inverse proportional with the window size of the black network. A larger window size will take more time to detect a packet loss. From the time between a packet gets lost, detected and recovery of the TCP stream at the black network, will allow more time for the red part of the diode to empty the buffer. This works conversely for the red network. A smaller window size result in a lower amount of buffered data.

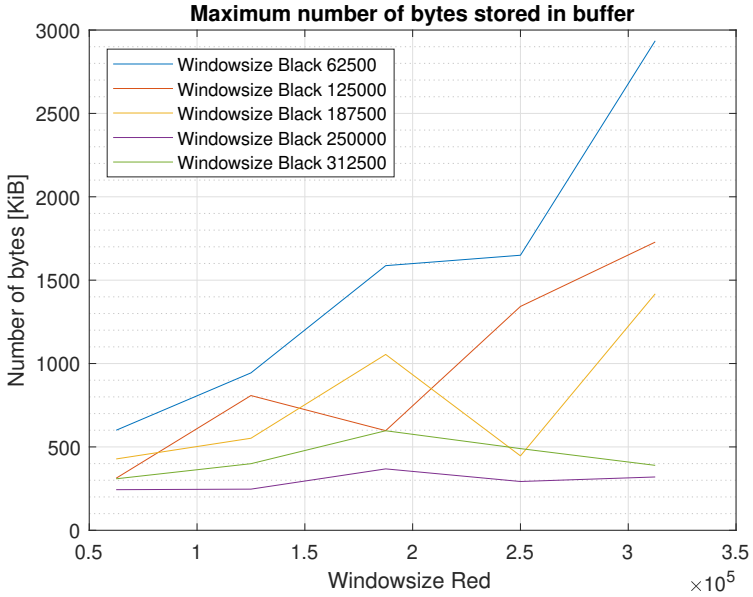


Figure 5.11: Comparison of different window sizes

The window size for the red TCP stream can not be controlled by the diode. However, the black window

size can be controlled by the diode. Therefore, the relationship between the round trip time and the window size of the black network will be researched. The simulation tests a situation where the red throughput is fixed and the window size and the round trip time will vary at the black network. The results of this are depicted in Figures 5.12 and 5.13. These figures present a clear linear correlation between the window sizes and the round trip time. From a RTT of at least $20ms$ (and a throughput of 100 Mbps), the maximum amount of data buffered is almost equal to the window size. This RTT allows the red network to discover a packet loss within one RTT, allowing the TCP stream to recover within a full window. Figure 5.12 will serve as recommendation for selecting a window size based on a known RTT.

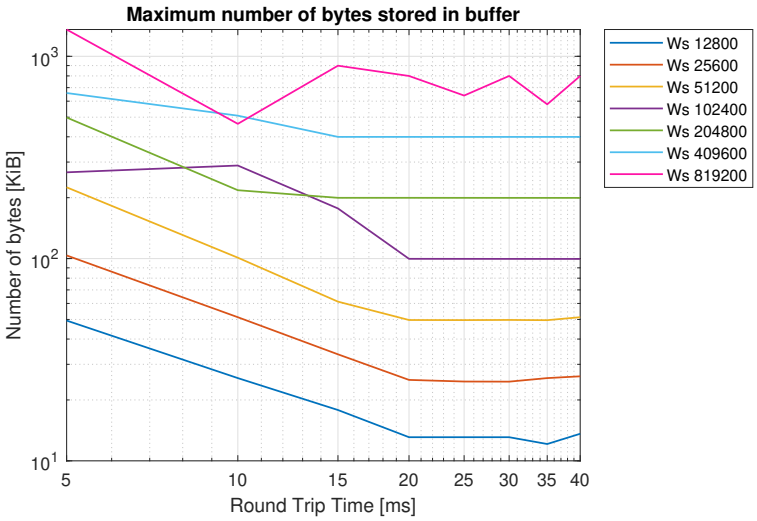


Figure 5.12: Maximum buffered data values for different window sizes

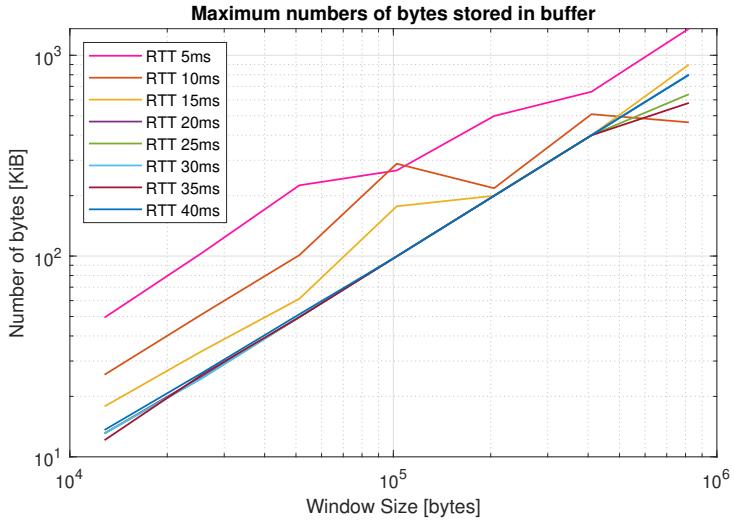


Figure 5.13: Maximum buffered data values for different round trip times

5.2. Configuration recommendations

The simulation allowed us to analyse the behaviour of the TCP stream under particular circumstances. The goal of the simulator is to find a set of configuration parameters to achieve the goals of the project (Section 1.3): minimise the buffer with a maximal performance. The list of configurable parameters for the TCP sockets in the data diode is presented in Figure 4.5. From the results of the simulations, a set of parameters can be constructed. We will discuss the parameters for each side of the network separately.

5.2.1. Black configuration

The diode is the receiver of the black TCP stream. The connection is initiated by the black source and the diode runs a passive process. The received packets should be properly acknowledged and passed through the diode to the buffer. In addition to this, should this part prevent the black throughput to exceed the red throughput. From the simulations we can conclude to not add features for recovery improvement. When an error happens at the black connection, the red connection gets some extra time to unload the buffer. Consecutive to the simulation results, we recommend the following configuration parameters:

- **No throughput limitation**
  The black part of the diode is not required to limit the throughput as a ratio of the red throughput. The average RTT of the red network is known to the network administrator. Adding throughput limitations adds complexity to the design of the diode and is not necessary.

- **No TCP Options**
  No TCP options will be implemented at the black part of the diode, except for the Window scale option. This window scale option allows the TCP socket to use a window size larger than can be defined by the standard field. The TCP SACK option has been tested and did not show any improvements. Comparing simulations with and without the use of the TCP SACK option, had similar results. Therefore, using the TCP SACK option does only add extra complexity to the implementation. The black TCP stream does not need any improvements with regards to the recovery time. A long recovery time in the black network does give some extra time for the red network to unload the buffer.

- **Simple acknowledgements**
  The TCP diode will use simple acknowledgements, instead of using delayed acknowledgements (RFC1122 [12]). Sending an acknowledgement for each received segment, will be executed simultaneously with passing the segment through the buffer. The black TCP stream could probably not achieve the most efficiency by this method, which is a good scenario for our system. The black part knows exactly what segment is passed to the diode and what not, as this variable is stored in the TCB. Furthermore, does this decrease the complexity of the implementation method.

- **Large window size**
  The black network will be configured with a relatively large window size. This will ensure the capability for a high throughput and is simple to implement, using the window scale option. Furthermore, a large window size is inherent to increasing packet loss detection time. If the black source does need more time to detect an error, the recovery time will also take more time. As discussed before, this is beneficial for the red TCP stream to unload the buffer.

5.2.2. Red configuration

The red part of the diode is responsible for transferring the segments out of the buffer, towards the red receiver. The diode should initiate and establish the TCP connection. It will manage the buffer and requires a segment-transfer managing process. The TCP stream needs a high efficiency, a high throughput and a small recovery time. Furthermore, the diode is should process the incoming acknowledgements for the receiver. This process should check if all segments arrived correctly, or if some packets are lost and need to be retransmitted. Proper acknowledged segments can be removed from the buffer, which should be notified to the segment-transfer manager. The transfer manager is also responsible for keeping track of the segments in transit and the current congestion window. The simulation results will recommend the following configuration for the red part of the diode:

- **No congestion avoidance algorithm**
  A congestion avoidance algorithm prevents congestion on the network and improves the TCP-friendliness of a TCP stream by distributing the network resources fairly. The algorithm is responsible for the size of the congestion window. It will "sense" the current available capacity of the network by approaching a fitting window size. If a packet loss occurs, the window will be reduced and the sensing will start again. The simulation results showed a peak in the buffer occurred due to this phenomenon. The red network is a trusted domain and in control of the administrator of the diode. Therefore, the congestion control could be disabled to solve the peak problem in the buffer. This will also decrease the complexity as no algorithm for congestion control needs be implemented.

- **Fast Recovery**
  By disabling the congestion control, the congestion window is also disabled. This will pose issues to the recovery of a lost packet. The congestion window should be reduced during the recovery period (Section 2.2.6). Otherwise, each lost packet will delay the recovery with a full RTT period and will decrease the efficiency of the throughput drastically. The TCP NewReno algorithm will be used to detect packet losses and improve recovery time.

- **No TCP options**
  The TCP SACK option is tested by the simulation and does not show improvements with regards to the amount of buffered data. Additional features requires more complexity for implementation. The tested TCP options did not provide significant benefits. Therefore, no TCP options will be implemented at the red part of the diode.

- **Small window size**
  A relative small window size is preferred at the red part the diode. Lost packets could be detected earlier and recovery time will be reduced. The red part aims to achieve a high performance with high efficiency. The simulation results shows a small window size will contribute to a higher efficiency.

5.3. Conclusion

To interpret the output from the tests performed in the previous chapter, we converted the data to values applicable to the parameters of Equation (3.6). In Chapter 5, we analysed the behaviour of the parameters to the amount of buffered data. The goal of the simulation was to find an optimal set of parameters to minimise the buffer space, while achieving high performance of the diode. We learned the negative effect of the congestion avoidance algorithm to the required buffer space. The algorithm can be neglected due to the controlled environment of the sending part of the diode. Furthermore, the window sizes on the black network should be relatively large to the red window size, if possible. The TCP SACK option, seemed very promising with regards to reducing the required buffer size (Appendix A). However, this option does improve the recovery time. Unfortunately, the peak of the buffer is not caused by the recovery of the lost packet. Instead, the recovery of a full throttle connection speed is much more determining. Without the need of *sensing* the capacity of the network resources, the TCP stream should be allowed to use the maximum after the lost packets is recovered. To **summarise** the simulation results:

- The TCP SACK option does not reduce the amount of buffered data in the long-term. The simulation runs with TCP SACK enabled did not show improvement when compared to the simulation runs with TCP SACK disabled.

- Any congestion avoidance algorithm will reduce the congestion window after a packet loss is detected. This will result in a peak in the buffered data. This part of the congestion algorithm can be disabled to avoid the peak in the buffer.

- With TCP disabled and the *sensing* part of the congestion avoidance algorithm disabled, Equation (3.2) still holds. However, the window size of the black TCP stream is preferred to be relatively large and the window size of the red TCP stream is preferred to be relatively small. A larger window size will increase the error detection time slightly. The black network will be temporarily delayed, which will save the red TCP stream some time to empty the buffer.

6

High-level hardware design

Goal 3 of this project is to develop a high-level hardware design which can be used to implement the proposed system in a hardware environment (Section 1.3). Implementing a network diode featuring TCP requires some challenges to be solved as was discussed in Chapter 2. When a network diode is installed between two network domains, the domains are completely separated. Only transfers in one direction can pass through the diode. TCP is a transmission protocol and is one of the most common protocols, used in many of current data transfers [42]. However, TCP requires an established two-way connection to transmit data. To ensure our design will be implemented in hardware and the data diode only allows traffic in one direction, the original TCP stream will be "duplicated". The original source will send the data, using TCP, to the part of the diode in the unsafe (black) network. The diode will transfer the *payload* from the unsafe part to the part of the diode which is connected to the trusted (red) network. At the red part of the diode, the original TCP stream will be reproduced to transfer the payload to the original receiver. To ensure no data is lost, the reliability of TCP will be used to ensure all packets will be sent correctly. TCP will retransmit packets when necessary. Therefore, the packets need to be available until they are acknowledged. The packets will be temporarily stored in a buffer. We want to know the minimum required size of this buffer, because the network diode (including the buffer) will be implemented in hardware.

In Chapter 3, we proposed a method to determine the required buffer size of the diode. This resulted in eqs. (3.1), (3.2) and (3.6). We defined a model of the system such that the amount of data in the buffer is determined by the data flow of both TCP streams, expressed in throughput. To avoid an infinite amount of buffered data, the throughput of the black TCP stream should be at most equal to the throughput of the red TCP stream. Consequently, we required a method to limit the black throughput, and therefore, we required a method to determine the red throughput. We formulated eq. (3.6) to find the significant factors for a TCP throughput. The behaviour of these variables is evaluated in Chapter 4. The analysis, based on simulation results, was discussed in Chapter 5. From the results, a set of recommended configuration parameters for each part of the network followed. The parameters configure the settings of the TCP stream to limit or improve the connection. The throughput, recovery time and required buffer space was taken in account.

In this chapter we will discuss the implementation of the functionality of the network diode. The design will have duplicate implementations for similar functions at both sides of the network, as well as unique implementations for side-specific processes. The hardware design will be presented as an accurate functional block diagram. The configuration recommendations from Sections 5.2.1 and 5.2.2 were taken into account during the development of the hardware design.

6.1. Overview

The high-level hardware design presented in this chapter, should function as the backbone of an implementation at hardware level. The design can be effortless converted to a VHDL project. The implementation is presented as a functional block diagram, in which each block represents a function or process and the lines represent a data- or control flow. The design will include the complete data- and control flow of packets handled by the data diode. Every process from the incoming packets to the transmission packets will be dis-

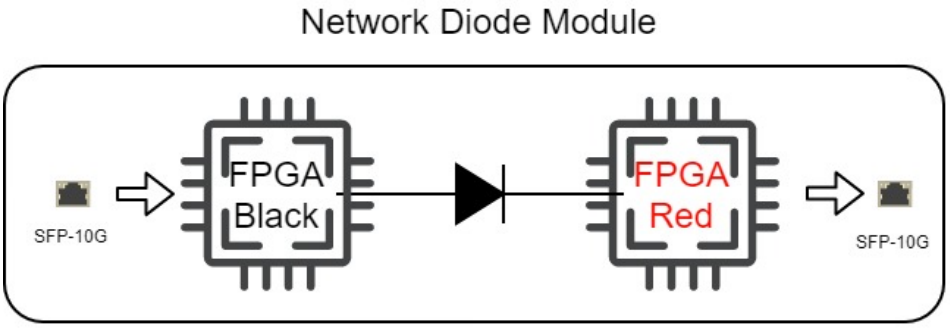


Figure 6.1: Top-level overview of the implementation of the data diode

cussed. The hardware design will present a modular design, allowing for more extensions or improvements in the future. All possible TCP features are considered based on the simulation results.

A top-level overview of the design implementation of the network diode is depicted in Figure 6.1. The network diode will have two network interfaces: one SFP-module at each side of the network. The transfer is started by the black source, and the packets will be received at the black side of the diode. The FPGA-black will process the incoming segments. It will respond with proper acknowledgements and transmits the segment via an electrical connection to the FPGA-red. The red FPGA will receive packets from the FPGA-black. It will store a copy of the segments in its buffer and will initiate the transfer to the red receiver. The red FPGA will manage the red TCP stream, including buffer management and packet retransmission.

A comprehensive description of the functional block diagram is depicted in Appendix D. The black hatched part includes all processes with packets in the the black network, the red hatched part is for the red network. The buffer is the only part connected to both parts of the diode and therefore not colour coded. The input connection is from the black part and the output is connected to the red part. With this implementation, the black part is not capable of knowing the saturation of the buffer. This is necessary to satisfy the diode's main function: full separation of network domains. By this method, the black part is not capable of getting any feedback of the status of the red network. The general processes for each side of the network are depicted in Figure 6.1 and separated for each side of the network. The black arrows represent the the data flow of the packets in the black network and the red arrows indicate the red data flow. The blue arrows represent the control flow in each side of the network. The yellow blocks indicate a hardware implementation, realised by combinational logic and registers. The purple blocks indicate a soft-core implementation, realised on a RISC-V core. The data flow of the TCP stream passes through only hardware blocks, to ensure the high data rate. Other processes will be implemented in soft-core. The RISC-V soft-core allows less complicated designs for more complex functions. However, this requires overhead from transferring the data to the soft-core, and vice versa, which will introduce extra delays. The buffer receives input from the black network and the output is connected to the red network. The network diode will have two Ethernet interfaces: one connected to the black network and one connected to the red network. The black interface will receive packets from the original source and will transmit acknowledgements over this interface. The red Ethernet interface will be used to transmit segments from the diode's buffer and receive acknowledgements. The processes in the diode will be discussed below.

The packets will enter the diode as a stream of bytes. To interpret the bytes, the stream is converted to an **AXI4-Stream** [5]. The AXI4-Stream protocol is a convenient interface to exchange data between components. The interface allows communication between a master, that generates data, to a slave, that receives data. The protocol allows both master and slave to agree when data can be transferred. The slave will notify the master it is ready to receive. When the data is marked valid by the master, the data will be transferred to the slave.

To explain the implementations of the functionality on the FPGA's, we will start to discuss the global operations of each side of the diode. Thereafter, we will elaborate the main function blocks in more detail.

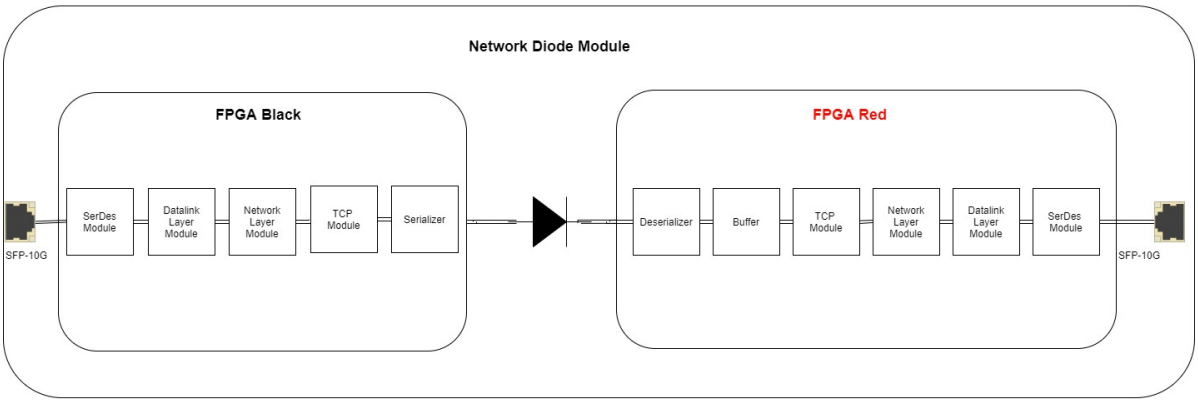


Figure 6.2: Top-level overview of the processes in the FPGA

Black FPGA processes

The black part of the network diode is the receiving TCP socket. The main function of this part is basically: receiving packets, passing segments through the diode (adding segments to the buffer), and replying with proper acknowledgements. The packets will enter the diode on an optical Ethernet interface. The optical interface represents the physical layer of the OSI-model. The SFP-module will convert the optical signal to electrical data. The serial data will be converted to parallel data (AXI-4 stream) by the SerDes module. The data will then flow through several functional blocks implemented in the FPGA. Each block executes some operations on the data and will pass it through to the next block. The direction of the data flow is is depicted in Figures 6.1 and 6.2. The explanation of the functional blocks will follow the direction of the data.

- **Data link layer module**
  The first process is handled by the data link layer (Ethernet). The packet will be checked for the Ethernet fields. The CRC will be calculated to detect any errors. The MAC addresses will be verified to ensure the source and destination devices are correct. Thereafter, the Ethertype will be checked, and the packet will be passed to the corresponding block. Our diode will be designed for the TCP/IP suite. Therefore, the packets from Ethertype IP will be passed to the next block. The other Ethertypes, ARP-requests for example, will be passed to an embedded soft-core processor. These packets do not contain critical data, but require complex functionality. However, transferring these packets to a softcore could introduce additional delay, but this does not effect the performance of the diode. The softcore will handle network packets and manage a network interface table.

  The data-link module processes packets the other direction likewise. It receives a packet from the network layer module or the softcore, and will attach an Ethernet header to it. The module creates an Ethernet frame, including the MAC addresses, and prepares the packet to be transmitted by the network interface.

  - **Input signals**
    - Ethernet-packet [AXI-4 stream]
  - **Output signals**
    - "Ethernet payload" [AXI-4 stream]
      The Ehternet payload contains the payload of the packet including the headers of IP and TCP.

- **Network layer module**
  The data diode will be designed to perform transfers using the TCP/IP suite. Therefore, the network layer module in our design only processes the Internet Protocol (IP). The incoming packets will be received from the data link module. The other protocols will not be passed to this module, but will be handled by a soft-core module instead. The IP module will process the incoming packets. The process will acquire the address fields from the IP-header to verify the source and destination of the packet. The header checksum will be verified to ensure the packet does not contain any errors. Furthermore, the protocol will be checked. All packets containing an other protocol than TCP will be dropped. At

his layer in the OSI-model only packets containing data payload will enter. Therefor, every packet not using TCP should not be handled by our diode. The network layer module will create the IP-header when transmitting packets. The header fields will be completed and the stream will be passed to the data link module.

  - **Input signals**
    - ◇ Ethernet-payload [AXI-4 stream]
  - **Output signals**
    - ◇ IPv4 header [logic vector]
      The IPv4 header contains data for the control flow of TCP
    - ◇ "IPv4 payload" [AXI-4 stream]
      The IPv4 payload contains the payload of the packet including the header of TCP.

- **Transport layer module**
  The transport layer module is the essential module of the the diode. The black FPGA will only contain a passive module. This side of the diode only needs to send replies to the black source. So the module will only process incoming packets. The packets will be received from the network module, which already stripped the IPv4 header of the packet. This header contains information that is required for validating the checksum of the TCP segment. The processes of the TCP module can be broken down to smaller functional blocks. First, the fields of the segment header will acquired, which is functionally almost the same as the other modules do. Thereafter, the information will be passed to a block that **verifies the TCP segment**. This block will checks the segment and validate whether segment is correct and checks if the diode is available to receive such a segment. Therefore, it requires information about the connection state and transmission state. These are acquired by the **TCB**. If the segment is not correct, the segment will be dropped and a corresponding segment will be prepared to reply if necessary. If the segment is correct, it will be passed to a block called: **process TCP segment**. This block will check all other fields of the segment according the RFC rules applied. These processes include verifying sequence and acknowledge numbers, update the FSM, update the transmission state values, prepare responds, etc. Furthermore, this block will transfer the data over the diode, via the serializer module. This is where the packet leaves the black network, and enters the diode connection.

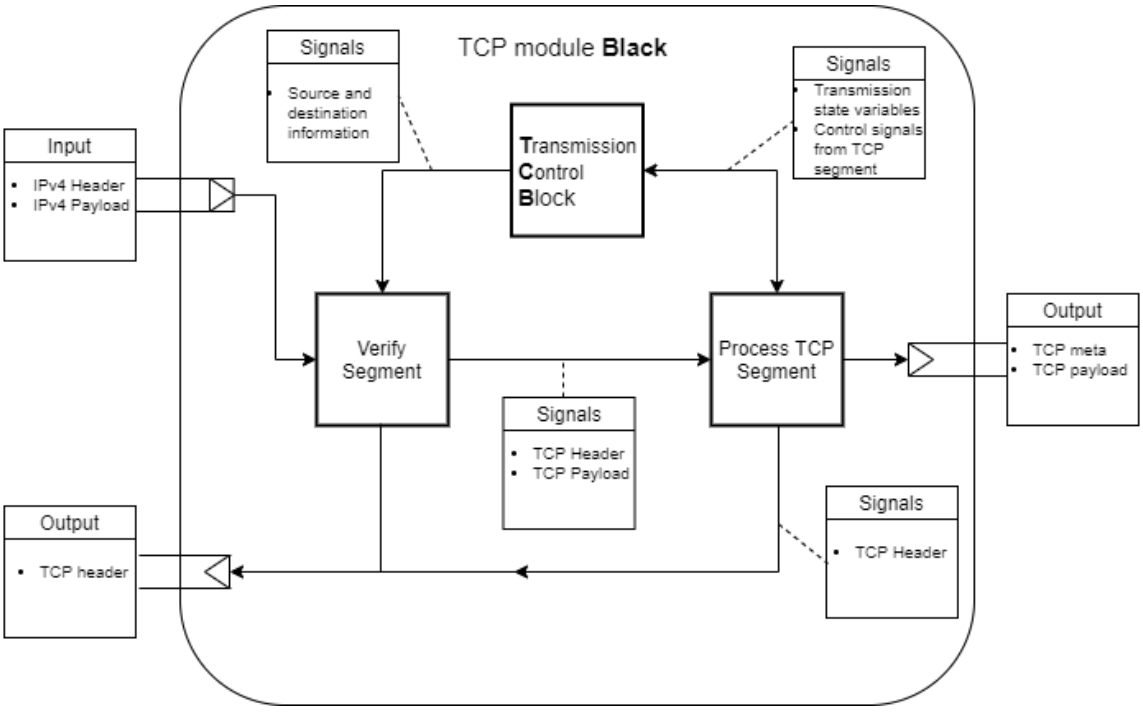


Figure 6.3: Overview of the red TCP module

The functional blocks of *TCP verification, segment processing and TCB* will be discussed in more detail later. The transport layer module creates acknowledging segments as a reply to the received segment. The segment header fields will be completed by information provided by the TCB. The segment will then be transmitted to the network layer module.

- **Input signals**
  - ◇ IPv4-payload [AXI-4 stream]
  - ◇ IPv4-header [logic vector]
- **Output signals**
  - ◇ TCP-header (to diode connection) [logic vector]
  - ◇ TCP-payload (to diode connection) [AXI-4 stream]

Red network processes

The red FPGA process packets in the red network. Actually two processes will be executed in this FPGA as the FPGA will have two incoming connections. The first process will receive TCP segments from the black FPGA via the diode link. These segments will be de-serialised into a parallel signal and will then be appended to the buffer. The FPGA will process the packet to finally transmit the packet to the SerDes module. The SerDes module will convert the parallel data into serial data for the SFP-module to transfer the packet to the original receiver. The second process will receive packets from the SFP-module, via the SerDes module. The packets will contain no data, but only control signals. These packets contain status information about the connection and will mostly consist of acknowledgements. The data flow in the red FPGA will start at the diode side. The packet will enter the the FPGA from there to the buffer. It will be processed by some blocks before it is transmitted. The segment will be removed from the buffer during the processing of the corresponding acknowledgement. Therefore, the red FPGA will contain the following functional blocks:

- **Buffer**
  The packet will be stripped down before it leaves the black FPGA. Only the TCP segment will enter the buffer. The buffer is only accessible in the red network. After the black FPGA has transferred the TCP segment over the diode, it does not need to store the segment. In the worst-case scenario, the acknowledgement did not receive the black source and the black source will retransmit the segment. The red FPGA, on the other side, does need to store the segment. This FPGA is responsible for transferring the segment to the red receiver. Until the segment is acknowledged by the receiver, the red FPGA is required to have a copy of the segment. Therefore, the buffer will be implemented on the red FPGA.

  - **Input signals**
    - ◇ TCP metadata (from diode connection) [logic vector]
    - ◇ TCP payload (from diode connection) [AXI-4 stream]
    - ◇ Transmission state variables (Red) [logic vector]
  - **Output signals**
    - ◇ TCP metadata [logic vector]
    - ◇ TCP payload [AXI-4-stream]

- **TCP Module**
  The TCP module on the red FPGA is responsible for transferring the packets from the buffer to the original receiver. Therefore, it has to process packets coming from two directions. First it has to retrieve the segment from the buffer and prepare it to transfer. Secondly, it has to process incoming packets, mainly consisting of acknowledgements. The incoming packets will be handled by the block: **process TCP segment**. This block will handle the control signals and connection information provided by the received segments. The acknowledgements will notify the buffer to remove the acknowledged segments. This information is also stored inside the **TCB** of the module. After a segment is removed from the buffer, the module will be able to send new data, as there is new room inside the transfer window. This window is managed by the **segment transmission manager**. The segment transmission manager is responsible for transferring the correct TCP segments. This includes processes like retransmissions and managing the connection.

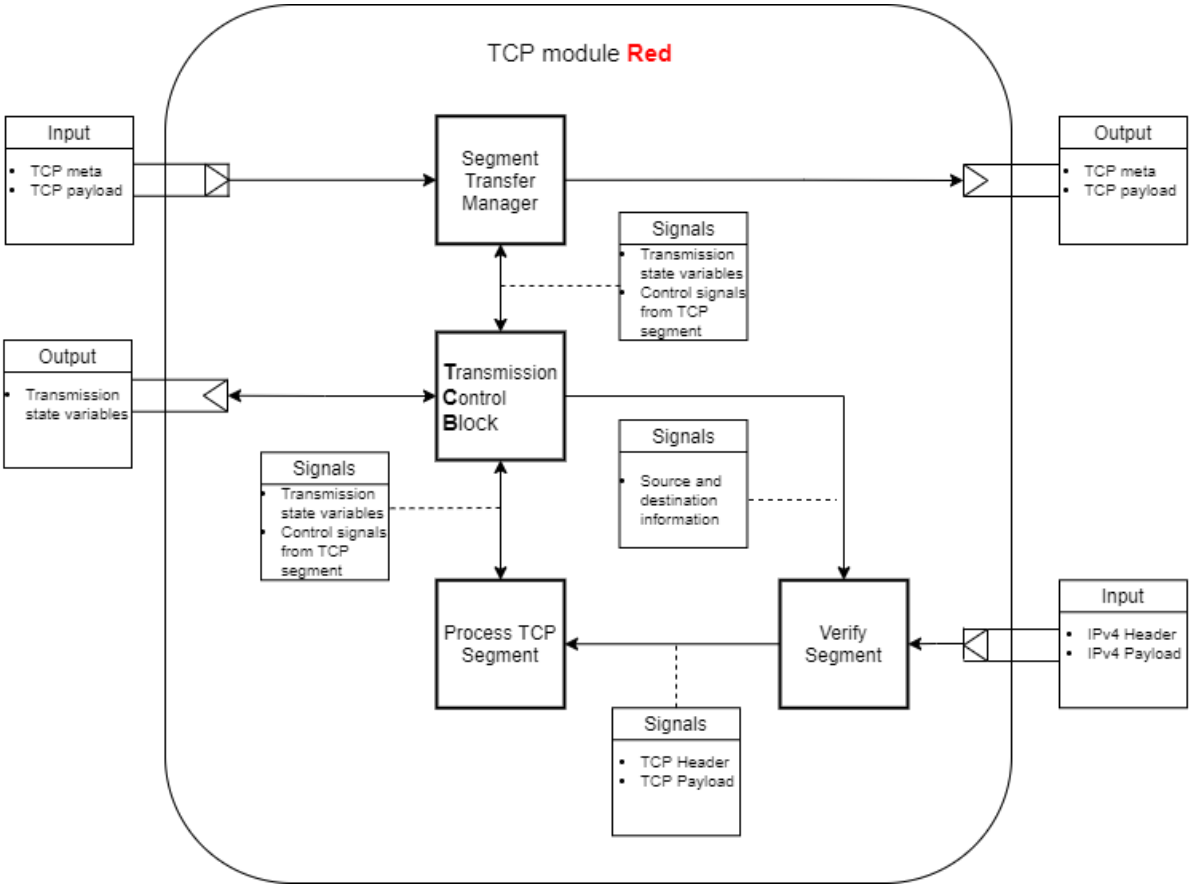


Figure 6.4: Overview of the red TCP module

The functional blocks of *TCP segment processing, the TCB and segment transmission manager* will be discussed in more detail in following sections. The transport layer module creates acknowledging segments as a reply to the received segment.The segment header fields will be completed by information provided by the TCB. The segment willthen be transmitted to the network layer module

- **Input signals**
  - Transmission flow
    - TCP metadata [logic vector]
    - TCP payload [AXI-4 stream]
  - Recipient flow
    - IPv4-payload [AXI-4 stream]
    - IPv4-header [logic vector]
- **Output signals**
  - Transmision flow
    - TCP Segment [AXI-4 stream]
      The TCP Segment does contain the TCP header and the TCP payload
  - Recipient flow
    - Transmission state variables [logic vector]

- **Network layer and datalink layer module**
  The network layer module and the datalink layer module for the black and the red FPGA will be the same. There will be no discrepancy between the functionality of these blocks. These modules will strip or add the header of the corresponding layer. The addresses of the recipient packets will be validated. Furthermore, a verification or calculation of the checksum of the packets will be executed, dependent on the direction of its current flow.

6.2. Verify TCP segment

The first field of the TCP header that will be checked is the checksum. The checksum is calculated using the TCP header and the pseudo-header from the IP-layer as was discussed in Section 2.2.2. Therefore, the information of the IPv4 header is required for this module. If the checksum is not correct, the packet will be dropped. Consequently the source and destination addresses and ports will be verified, which identify the connection. Furthermore, the connection state (see Figure 2.6) will be checked, whether the diode is able to receive packets for this connection. This information is acquired from the TCB. If the connection is in the *Closed* state, the packet is dropped and a proper respond will be prepared. The responding packet will contain a *RST* signal and is created conform RFC 793 [55]. The module will send the created TCP segment to the network layer module. If the connection is in another state than *Closed*, the TCP header and payload will be forwarded to the *process TCP segment* block.

- **Input signals**
  - IPv4-header [logic vector]
  - TCP-header [logic vector]
  - TCP-payload [logic vector]
  - Source/destination data [logic vector]
  - FSM-state [logic vector]
- **Output signals**
  - TCP-header [logic vector]
  - TCP-payload [logic vector]

6.3. Process TCP Segment

After the segment is verified by the previous block, the segment is ready to be processed. The control bits and signals are extracted from the header field and the current state of the connection. This process determines the response, if applicable, to the received segment. If an acknowledgement or a reset signal is required, this process creates the proper segment to send. The payload of the segment will only be processed if all of the other fields of the segment are correct. Multiple checks will be executed to verify the acceptability of the segment. First, the state of the connection will be checked, again.

If the connection is in the *Listen* state, an appropriate *SYN* segment conform the rules of RFC793 is expected. Otherwise, the packet should be dropped and a *RST* segment will be sent. If a correct SYN segment is received, then the black source is establishing the connection and this is the first message of the 3-way handshake. The module will extract the important values and will send the values to the TCB to update the *transmission state variables*. These contain values for the transmission state pointers, TCP options and a notification for the FSM to move to the next state: *SYN received*. This state will only apply for the black FPGA, because the black part will wait for the original source to initiate a connection.

If the connection is in the *SYN send* state, the module is expecting a SYN for the negotiation of the 3-way handshake. This will only occur in the red FPGA, because this state is reached by the sender of the TCP session. After the segment arrives, first the acknowledgement number is verified. The acknowledgement number should be in between the initial sequence number and the next-to-send sequence number. If this is not the case, the segment should be dropped and a RST should be send where appropriate. The same check applies for the sequence number if the segment does include a SYN. This is the moment of the negotiation. Therefore, the TCP options are being exchanged to agree on the TCP session configurations. The available TCP options configurations for our design are discussed in Section 5.1. After the SYN is processed, the TCB should be notified to go the next state, dependent on the presence of the ACK bit: *SYN received or Established*.

If the connection is not in the Closed, Listen or SYN send state. The segment is processed in a 6-step cycle. This cycle is extracted from the 8-step cycle from RFC793 with two steps removed: the security and precedence check and the URG-bit check are removed. Both will not be used in our design. The security and precedence values should be provided by the IP layer and the URG bit is used for other applications than just raw data transfers. Each step in the cycle is a check on one of the control bits or fields of the segment. The control signals determine the next process for the segment and will manage the connection state.

Segment length	Receive window	Check
0	0	SEG.SEQ = RCV.NXT
0	>0	RCV.NXT =<SEG.SEQ <RCV.NXT + RCV.WND
>0	0	not acceptable
>0	>0	RCV.NXT =<SEG.SEQ <RCV.NXT + RCV.WND or RCV.NXT =<SEG.SEQ + SEG.LEN-1 <RCV.NXT + RCV.WND

Table 6.1: The possible conditions for the TCP segment to be acceptable

1. **Verify segment's acceptability**
   The first check is to validate whether the incoming segment is acceptable based on the sequence number and the current size of the receiving window of the module. The valid conditions for a segment to be acceptable is presented in Table 6.1.

   If the receive window is zero, no segments will be accepted. Only exceptions are segments containing a valid ACK or RST bit. Unacceptable segments should be answered with an acknowledgement, unless the RST bit is set. After a acknowledgement is transmitted, the unacceptable segment should be dropped. If the RST bit is set, no acknowledgement should be sent and the unacceptable segment should be dropped immediately. Only segments with a sequence number equal to the RCV.NXT pointer value will be accepted at the black FGPA. The black FPGA will have no buffer and therefore it is unable to reorder received segments. If the set of rules for this step applies to the segment, the following check will be done.

2. **Check the RST bit**
   A RST bit indicates that the receiver should delete the connection without furthermore communication. An error has occurred somewhere in the connection and the sender of the RST bit will break down the connection. If the RST bit is valid, the TCB values of this connection will be emptied and the connection will be terminated. Whether the RST bit is valid, depends on the current state of the connection.

   If the connection is in the *SYN received* state, the TCP module expects an ACK to establish the connection. In this state, the black FPGA will just drop the segment and will return to the *Listen* state. If the red FPGA receives a RST bit in this state, the connection is refused by the red receiver. Therefore, the TCB values should be emptied and a hard reset for the connection is required. The red FPGA should transit to the *Closed* state.

   If the segment reached this point and the connection is in another state then the *SYN received* state, each outstanding segment should receive a reset signal. Therefore, all segment queues and all TCB values should be emptied. The connection should transit to the *Closed* state.

3. **Check SYN bit**
   If the segment reached this step and there is a SYN bit in the segment, than the segment is an error. The connection should be reset. The segment queues and the TCB values should be emptied and the connection should transit to the *Closed* state.

4. **Check acknowledgement fields**
   First, the segment should be checked if the ACK bit is present. If the ACK bit is absent, the segment should be dropped. If the ACK bit is present, check the state. If the connection is in the *SYN received* state, the acknowledgement number of the segment will be checked. The acknowledgement number should be in between SND.UNA and SND.NXT to be valid. If the segment is valid, than the connection should transition to the *Established* state and the segment should be processed the same as in the other states. If the segment's acknowledgement number is not valid, a RST should be sent instead. Consequently the segments acknowledgement number is validated. When the acknowledgement number is larger (or equal) to the highest unacknowledged yet transmitted segment, the received segment indicates a duplicate acknowledgement. This should be notified to the TCB and add up to the duplicate ACK counter. Furthermore, the acknowledgement number should be in between SND.UNA and the largest sequence number transmitted. Otherwise the acknowledgement is invalid. If there were unacknowledged segments in the buffer, the segments with a sequence number smaller or equal than the received segment's acknowledgement number can be removed from the buffer and the transmission

state variables should be updated in the TCB. Furthermore, if this was a new acknowledgement, values from the window size and TCP options are updated. The next state is determined by the TCB and is dependent on the current state of the connection. On the black FPGA, if a FIN bit is acknowledged, the connection should close. On the red FPGA, if a FIN bit of our module is acknowledged, the connection should transition to the *FIN-wait-2* or *Time-wait* state.

5. **Process segment payload**
   If the segment contains a payload with size larger than zero, the connection should be in the *Established* or a *FIN-wait* state. If the connection is in another state the payload should be ignored. The transmission state variables will be updated to the TCB. the RCV.NXT pointer is set to the value of the lenght of the segment added to the segment's sequence number. Normally the window size should be recalculated at this moment. In our design, the window sizes are of a fixed size and no recalculation will be executed. The black FPGA will also transmit the payload together with the TCP header to the diode connection. If this is done, the segment will continue to be processed by the last step.

6. **Check the FIN bit**
   This step can only be executed when the connection is not in the *Closed*, *Listen* or *SYN sent* state, because the sequence number of the segment can not be validate in these states. The segment should be ignored and dropped. If the FIN bit is present in the segment, verify the sequence number. An invalid FIN should be dropped. A valid FIN bit should update the RCV.NXT pointer value and an acknowledgement should be sent. Depending on the current, should the TCB transit the state of the connection. The black FPGA can prepare to terminate the connection and empty the TCB. The red FPGA should be finished and this FIN is the indication that the receiver is also finished.

Difference between red and black

Both FPGA's will include this model, because both will receive TCP segments from the external connection. The black FPGA is the receiving side of the TCP connection. Therefore, only the black FPGA should receive segments containing a payload. The payload contains the data the user wants to transfer over the diode. The other segments that will arrive at the black FPGA are control signals to negotiate, initiate or terminate the connection (parameters). The black TCP module will execute a *passive open* and therefore start in the *Listen* state. During the connection, it will only accept segments that are consecutive. The black FPGA does not include a buffer, therefore it is unable to reorder any segments. It is able to pass the out-of-order segment over the diode connection already, which we will discuss in Chapter 7. If a valid segment arrived in the right order, the black FPGA will prepare an acknowledgement and will simultaneously send the data over the diode. The connection is terminated if the black source has finished sending all the data and transmitted a TCP FIN segment. The black FPGA will not acknowledge the FIN segment until it received all segments. Therefore, when acknowledging the FIN signal, it is finished with its tasks and the FIN-wait-1 state can be omitted.

The red part on the other hand is the initiator of the TCP stream. The red FPGA will start the connection in the *Closed* state. When the first segment arrives at the buffer, the TCP module will do an *active open* and should transmit the first SYN segment. The other control signal will arrive from the red receiver. The red FPGA should only receive controlling signals without a payload. It represents the sending side of the diode and has only data to send. Payload received from the red receiver will be dropped immediately. The process-TCP-segment module will mainly accept acknowledgements to update the TCB values transmission state variables. The red FPGA does include the buffer of the diode. Therefore, this module will notify the buffer to remove data which corresponds with the valid acknowledgements received from the original receiver.

- **Input signals**
  - IPv4-header [logic vector]
  - TCP-header [logic vector]
  - TCP-payload [logic vector]
  - Source/destination data [logic vector]
  - FSM-state [logic vector]
- **Output signals**
  - TCP-header [logic vector]
  - TCP-payload [logic vector]
  - Transmission state variables [logic vector]

6.4. Transmission Control Block

The Transmission Control Block (TCB) keeps track of all values for the connection. The TCB manages the values for the transmission state pointers and execute the TCP-state FSM based on the control flags supplied. These values are used in the protocol for many processes. These processes include verifying if a segment is valid and/or acceptable based on the control fields in the segment and processes that keeps track of the current state of the connection. If a module requires the value of specific transmission state values, it retrieves these from the TCB. Only the TCB is allowed to adjust the values. It will modify the values based on the signals retrieved from the TCP segments. It will also cycle between the TCP states when notified by a module. It will only send a value if it is valid. Therefore it is necessary only one module has the right to adjust these and keeps track of them. The values maintained by the TCB contribute to the mechanisms of TCP. Therefore, the following parameters will be stored in the TCB:

- **Source information data**, consisting of the IPv4 address and the TCP port
- **Destination information data**, consisting of the IPv4 address and the TCP port
- **RCV.WND** representing the most current advertised window size of the receiving device
- **RCV.UP** representing the urgent pointer of received segments
- **RCV.NXT**, the receive next pointer. The first expected segment should have a sequence number equal to this value.
- **IRS**, the Intial Receive Sequence number. This value will be used as reference value for the first received segment of the connection.
- **SND.WND**, representing the window size of this device.
- **SND.UP**, representing the urgent pointer of segments in the sending queue.
- **SND.WL1**, representing the sequence number of the segment which is used for the last window (SND.WND) update.
- **SND.WL2**, representing the acknowledgement number of the segment which is used for the last window (SND.WND) update
- **SND.MSS**, representing the value of the Maximum Segment Size used for this connection. This value is negotiated during the 3-way handshake
- **SND.WND_SCALE**, representing the value of the Window Scale from the last update (SND.WL2)
- **ISS**, the Initial Send Sequence number. The first sent segment will have this value and will be used as reference value.
- **FSM-state**, the FSM state is managed by the TCB.
- **DUP.ACK**, represents the duplicate acknowledgement counter.
- **Input signals**
  - Control signals [logic vector]
- **Output signals**
  - Source/destination data [logic vector]
  - Transmission state variables [logic vector]
  - FSM-state [logic vector]

6.5. Segment transmission manager

Only the red FPGA will have the segment transmission manager module as was discussed in Section 6.1. This module is responsible for the transmission of the diode data. Only the the red FPGA will transmit actual data. The black FPGA will only transmit TCP segments containing control signals, consisting of acknowledgements in particular. The black FPGA will only transmit a segment in reply to an incoming segment. Before any segment is transmitted, this module will check the current connection state. The red FPGA will start in the *Closed* state. If the first segment arrived on the buffer and this segment contains a SYN bit, it will initiate the TCP connection in the red network. Otherwise it will only send segments containing payload in the *Established* state. If the last segment in the buffer contains a FIN segment, this will only be sent if all previous segments have been successfully transmitted. Therefore, all the previous segments should have been acknowledged before the last segment containing the FIN bit will be sent. If this segment contains a payload, the payload will be sent in a separate segment. The other states will be cycled through be control segments received from the red receiver and the TCB will handle these. The red FPGA will active create TCP segments to transfer the data. The payload of the segments is received from the buffer. To determine which packets it will send, it uses the sliding window system of TCP. The segment transmission manager will cooperate with the TCB. The SND.NXT value represent the sequence number of the segment that should be send next. If $SEG.SEQ + SEG.LEN =< SND.UNA + SND.WND$, the segment is acceptable to send. In other words, if the length of the segment added to the sequence number does not exceed the right edge of the window, the segment is acceptable to send. There should be space left in the window to send a new segment. While the segment is transmitted, the segment transmission manager will start the RTO timer. If a RTO occurs for a segment, this module will set the value of SND.NXT back to the segment that is retransmitted and notifies this to the TCB. The fast recovery algorithm will adjust the congestion window until the connection is recovered. An equivalent process will be executed when a triple duplicate acknowledgement is received. For each duplicate acknowledgement the counter will be incremented. When three duplicate acknowledgements are received, the SND.NXT value will be set back.

- **Input signals**
  - TCP-header [logic vector]
  - TCP-payload [logic vector]
  - Source and destination information [logic vector]
  - Transmission state variables [logic vector]
  - FSM-state [logic vector]
- **Output signals**
  - TCP segment [AXI-4 Stream]
  - Control signals [logic vector]

6.6. Data and control flow

The processes on the segments can be separated into two different flow: the data flow and the control flow. The data flow is defined by the path which the data has to move along when it is transferred from the black source to red receiver trough the data diode. The data will enter this path at the black FPGA. The data will be structured as a stream. Each block the data stream passes, will execute a process on the stream. They will extract, strip, analyse or copy some of the parts of the data stream. First the headers will be split and analyses if the packet has the correct destination and protocol number. The checksum will be verified to ensure the packet does not contain an error. Both data flows are depicted in Figures 6.5 and 6.6 Finally, in the black FPGA, the control signals will be extracted from the TCP segment. These will be handled by the control the flow. The data flow continues to send the segment to the red part of the diode and to create the acknowledgement. This will go back through the modules which will add an extra header to the packet, which is similar to attach extra data to the stream. Therefore, the black data flow is circular. The data will enter the FPGA and, inside the black FPGA, the data flow will exit when the acknowledging segment is sent.

The red FPGA does have two data flows, because it has two input connections. The first data flow starts at the buffer. The data will flow from the buffer to the SFP-module to be transferred to the red receiver. The data from the buffer will be transformed into a data stream. Each block will add a header of the corresponding

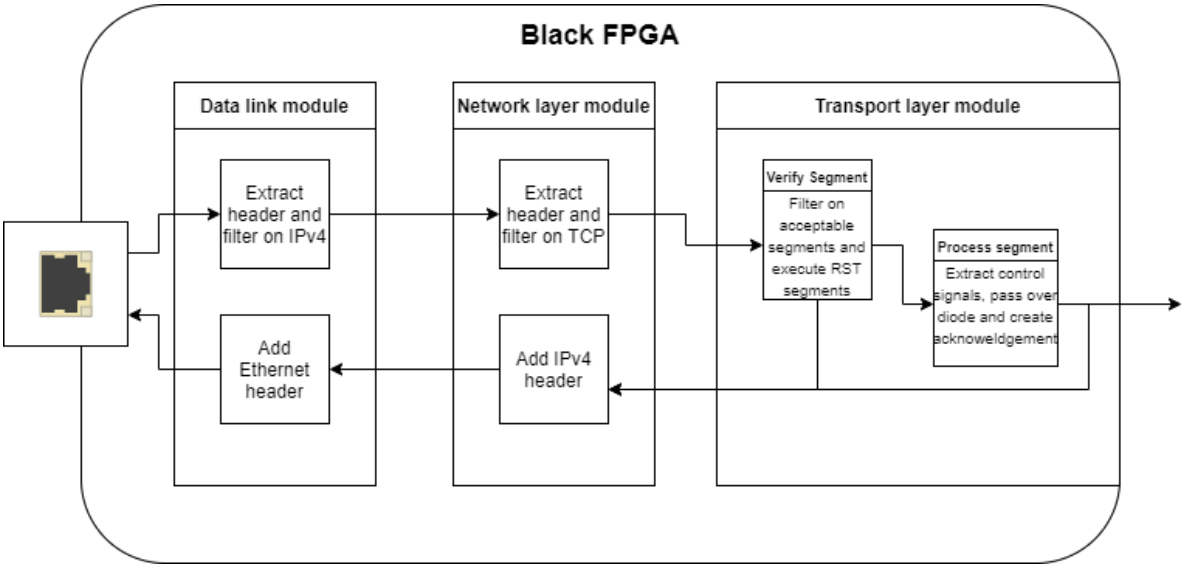


Figure 6.5: The data flow path of a packet in the black FPGA. The blocks annotate the action performed on the packet. The data flow is circular on this side

layer to the packet. The other data flow contains the receiving segments of the red receiver. This data flow is similar to the data flow of the black network. Only instead of sending an acknowledgement segment, this flow receive acknowledgement segments. These are passed to the TCP module, if it has the appropriate meta data, and will be analysed for control signals.

Each process acting on the data flow will be implemented in logic hardware. This should ensure the capacity to reach a high data rate. The data stream used in our design an AXI-4 stream. This protocol allows to create *packets* which will process multiple bits in one clock cycle. To achieve a data rate of 10 Gbps, a packet consisting of 64 bits could be used with a clock of 156.25 MHz.

All other processes not acting on the data will be defined as the control flow. These could be separated in two different flows. One flow will handle all packets which does not directly contribute to the data transfer over the diode. These are packets containing other protocols than IP/TCP. ARP or DHCP for example. There is no requirement to process these packets fast or at a high data rate, because these are mainly packets for networking. Therefore, these packets will be handled by a soft-core solution on the FPGA. When a packet with a different protocol is detected (and validated to be required for the network link), it will be transmitted to the soft-core which will process them. Therefore, the soft-core will maintain an ARP- and an interface table.

The other control flow is defined within the TCP module. The TCP module will extract and analyse the control bits and fields in the TCP segment's header. The signals will be transmitted to the TCB. The TCB will use these to update the transmission state variables and to cycle through the FSM of the TCP. The black FPGA requires the new values of the transmission state variables and the FSM state to determine what reply segment should contain. The red FPGA requires the updated values to determine what segment to send next. Furthermore, the buffer requires the values to remove the corresponding segments.

6.7. Conclusion

The last goal of this project is to develop a hardware design on a high-level. The results from previous chapter will form the basis of the configuration of the hardware design. In Chapter 6, a high-level hardware design was presented for implementing a network diode, featuring TCP, in hardware. The implementation will consist of two FPGAs, one for each side of the diode. The black FPGA and the red FPGA are connected by an electrical connection. This is connection is purposely used in one direction. This connection features the diode and separates both networks.

The black FPGA executes a passive process. It waits until a packet arrives and will process it. The packet will go through some functional blocks. Each block represents the processes of a different OSI-layer. The most significant block in our design is the TCP block. The processes acting on the TCP segments are separated in blocks. The TCP module on the black FPGA does contain the same blocks as the red FPGA. Only the red FPGA

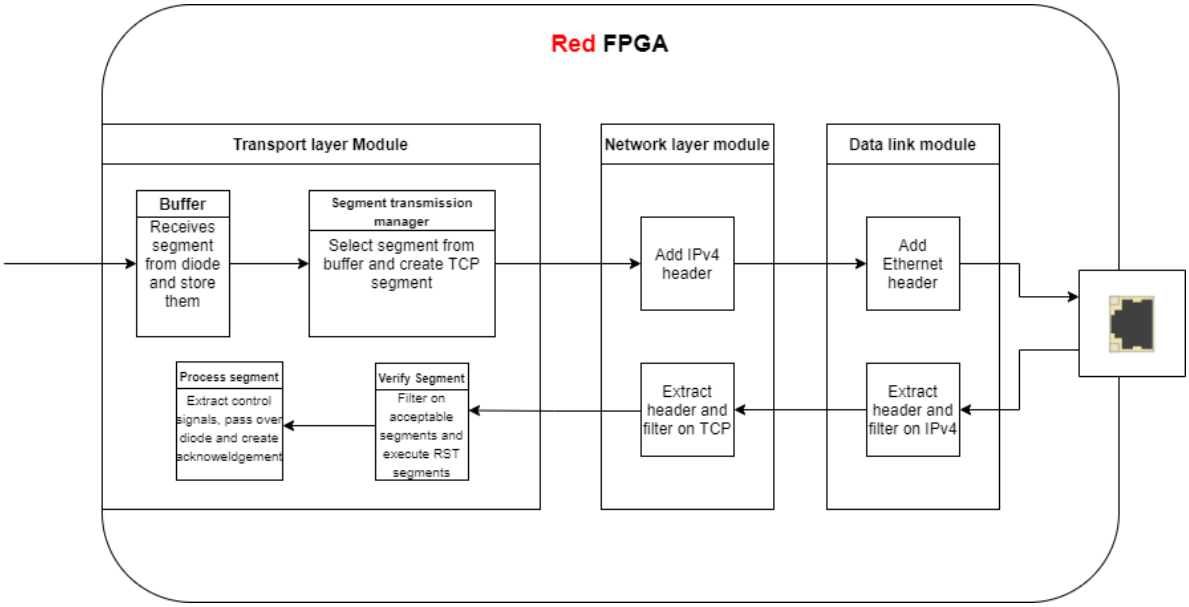


Figure 6.6: The data flow path of a packet in the red FPGA. The blocks annotate the action performed on the packet. There are two separate data flows on this side.

has one extra, because that is the active FPGA. If a packet arrives at the black FPGA, the control signals will be analysed to maintain the connection. This is done inside the TCP module. Finally, the segment will be transmitted over the diode connection, and an acknowledgement segment will be replied to the black source.

The red FPGA executes an active process. The red FPGA does have 2 main differences compared to the black FPGA: 1) it has a buffer 2) the TCP module does contain an extra block. The buffer features, together with the electrical connection between the FPGAs, the diode functionality. This is the border between the black and the red network domain. When the segment is sent by the black FPGA to the buffer it is out of control and it is up to the red FPGA to transfer the packet to the original receiver. The buffer is emptied by the TCP module of the red FPGA. This module does contain an extra process block: the *segment transfer manager*. This block determines when a segment is transmitted from the buffer. The segments will be removed from the buffer by the *process segment* block. This block will process the acknowledgements received from the red receiver. The black FPGA does contain the same block in the TCP module. Only there, the block will create an acknowledgement and will transmit the segment over the diode connection.

The processes acting upon the data transfer will all be implemented in logic hardware. This will ensure the capacity to achieve a high data rate. The AXI-4 stream protocol will be used. This allows 64 bits to be processed in one clock cycle. A clock of 156.25 MHz should then realise a throughput of 10 Gbps. The TCP module is designed modular. Therefore, the module allows a developer to implement extra features or TCP options in the future.

7

Conclusion

This chapter describes the conclusions that have followed from each step of the methodology of this project, represented by the chapters of this thesis. Furthermore, the considerations of this thesis are being discussed and recommendations for future work will be presented.

7.1. Conclusions

Based on the problem statement of this thesis: "To what extent is it possible to implement a network diode on an FPGA under realistic network environments, using the Transmission Control Protocol", we researched the current diode products on the market, which was described in Section 2.1. From the developments in the network security industry, we learned the bottleneck of this project, namely, TCP is a bidirectional flow, which has to be converted to a unidirectional flow. The standard solutions use PC's, which run proxy software to solve this issue. The only hardware implementation is the one-way diode connection, which does not cover the features of the TCP. The data flow is converted from bidirectional to unidirectional at the sending side of the diode. The information is sent using a custom-made protocol designed for the user. At the receiving side, the information is converted back to the bidirectional TCP flow. This has the advantage of an adjustable diode protocol. The functionality can be modified by updating the proxy PC's. Although the disadvantage of this is the high cost of the diode and the maintenance of those proxy PC's.

From the operating mechanisms of TCP described in Section 2.2 we can conclude it is a very reliable protocol. It has many features to establish the connection, manage the connection while taking in account other traffic and has enough room to implement new features for improvement. But those characteristics for TCP are only valid when communication is available both ways. A bi-directional connection is vital to the TCP methodology. The sending side of the connection requires feedback from the receiving side to operate. There is no adaption of TCP with only one-way traffic allowed. TCP implements algorithms to adapt to the current state of the network, which is mainly detected by packets loss or fluctuations. Those algorithms provide heuristic solutions to recover the connection, while achieving high efficiency and taking in account a fair share of the network resources.

With this knowledge, we continued to find a method to define a simplified model of our system for analysis purposes. In Chapter 3, a suitable model was proposed for system evaluation purposes. The conversion of TCP to one-way traffic (and reversed) will be processed on the same device as the diode is implemented. Together with the basic operations of TCP described in Chapter 2, and taking in account goal 1 (Section 1.3), a simplified model is presented in Section 3.1. This model consists of two separate TCP streams on both sides of the diode. The unsafe network is referred to as 'the black network' and the trusted network is referred to as 'the red network'. Because no feedback is possible, the two networks would operate separated. The only shared resource is the buffer. This buffer is filled with segments from the black network and emptied by the red network. The amount of data inside the buffer is almost equal to the window size of the red network and difference in throughput of both networks, as stated in Section 3.1.1. Therefore, the throughput from the black network must not exceed the red network. To determine the throughput for a TCP stream, a formula is proposed in Section 3.1.2, which consist of four parameters. In Sections 3.1.3 and 3.1.4 is discussed which parameters can be controlled on the diode for each side and to which parameters are determined by

the network.

To evaluate this set of parameters, a network simulation will be used. To select the most suitable simulation software, several network simulation solutions were discussed in Section 3.2. Commonly, two different types of simulators are considered: A Discrete Event Simulator (DES) or a network emulator. We tested three potential simulators: Mininet; NS-3; OMNeT++. The emulator provided by MiniNet, lacked the capacity to simulate high-speed transfers or process packets in a short time, because an emulator simulates real-time network traffic and requires sufficient resources to do the calculations, which our computers lacked. Both NS-3 and OMNeT++ are DESes and have implemented options to simulate TCP streams. NS-3 models networks by creating a network of nodes and corresponding connections. Network devices can be installed on those nodes to create a real network. Applications can be assigned to the network devices, which includes TCP sockets. The project needs to be compiled by an build system called *Waf*. OMNeT++ uses simpler block structures to design a model. It makes use of simple modules, which could be used for sending any kind of message. It is provided with an user-friendly GUI, which makes designing more comfortable for a beginner. OMNeT++ version 5 or higher is provided with the INET framework, which consists of common network components, including TCP. In addition to this, OMNeT++ IDE has an integrated analysis tool for the output statistics. Therefore, Omnet++ is selected as best simulation tool.

The set of parameters in Equation (3.6), which was discussed in Chapter 3, has to be simulated in OMNeT++. Therefore, the model described in Chapter 3 should be implemented in the simulation software. In Chapter 4, the TCP diode module for system simulation was developed. The model can enable all potential features of the network diode as well as the properties of a TCP connection. The network diode module is implemented using the INET framework of Omnet++ as described in Section 4.1. The standard host module, which is a module provided by the INET framework, will run the TCP Diode Application. This application is bound to two TCP sockets: one socket in the black network and one socket in the red network. The application copies all messages from the black socket to the red network. From the statistic records of the sockets, the value of buffered data is acquired. This value will be investigated during the simulations, according to the equations formulated in Chapter 3 (Equations (3.1), (3.2) and (3.6)). The parameters, as discussed in Sections 3.1.3 and 3.1.4, can all be configured in the simulator. First of all, Equation (3.2) has been tested, by researching the relationship between the throughput of the black and the red network with regards to the utilised buffer space. Thereafter, the TCP SACK option was analysed, after our hypothesis of the improvements for the buffer of the data diode (Appendix A). The influence of a congestion algorithm is questioned to be of a negative impact on the required buffer space. Therefore, tests without a congestion algorithm were performed. Finally, a relationship between the window sizes of the red and black network has been researched to validate Equation (3.1).

To interpret the output from the tests performed in the previous chapter, we converted the data to values applicable to the parameters of Equation (3.6). In Chapter 5, we analysed the behaviour of the parameters to the amount of buffered data. The goal of the simulation was to find an optimal set of parameters to minimise the buffer space, while achieving high performance of the diode. We learned the negative effect of the congestion avoidance algorithm to the required buffer space. The algorithm can be neglected due to the controlled environment of the sending part of the diode. Furthermore, the window sizes on the black network should be relatively large compared to the red window size, if possible. The TCP SACK option, seemed very promising with regards to reducing the required buffer size (Appendix A). However, this option does improve the recovery time. Unfortunately, the peak of the buffer is not caused by the recovery of the lost packet. Instead, the recovery of a full throttle connection speed is much more determining. Without the need of *sensing* the capacity of the network resources, the TCP stream should be allowed to use the maximum share of resources after the lost packets is recovered.

The last goal of this project is to develop a hardware design on a high-level. The results from the previous chapter will form the basis of the configuration of the hardware design. In Chapter 6, a high-level hardware design was presented for implementing a network diode, featuring TCP, in hardware. The implementation will consist of two FPGAs, one for each side of the diode. The black FPGA and the red FPGA are connected by an electrical connection. This connection is purposely used in one direction. This connection features the diode and separates both networks.

The black FPGA executes a passive process. It waits until a packet arrives and will process it. The packet will go through some functional blocks. Each block represents the processes of a different OSI-layer. The

most significant block in our design is the TCP block. The processes acting on the TCP segments are broken down in separate blocks. The TCP module on the black FPGA does contain the same blocks as the red FPGA. Only the red FPGA has one extra block, because that is the active FPGA. If a packet arrives at the black FPGA, the control signals will be analysed to maintain the connection. This is done inside the TCP module. Finally, the segment will be transmitted over the diode connection, and an acknowledgement segment will be replied to the black source.

The red FPGA executes an active process. The red FPGA does have two main differences compared to the black FPGA: 1) it has a buffer 2) the TCP module does contain an extra block. The buffer features, together with the electrical connection between the FPGAs, the diode functionality. This is the border between the black and the red network domain. When the segment is sent by the black FPGA to the buffer it is out of control and it is up to the red FPGA to transfer the packet to the original receiver. The buffer is emptied by the TCP module of the red FPGA. This module does contain an extra process block: the *segment transfer manager*. This block determines when a segment is transmitted from the buffer. The segments will be removed from the buffer by the *process segment* block. This block will process the acknowledgements received from the red receiver. The black FPGA does contain the same block in the TCP module. Only there, the block will create an acknowledgement and will transmit the segment over the diode connection.

The processes acting upon the data transfer will all be implemented in logic hardware. This will ensure the capacity to achieve a high data rate. The AXI-4 stream protocol will be used. This allows 64 bits to be processed in one clock cycle. A clock of 156.25 MHz should then realise a throughput of 10 Gbps. The TCP module is designed modular. Therefore, the module allows a developer to implement extra features or TCP options in the future.

7.2. Main contributions

The models, simulation results and the high-level hardware design developed during this project contribute to answer the main research question:

To what extent is it possible to implement a network diode on an FPGA under realistic network environments, using the Transmission Control Protocol?

The high-level hardware design presented in Chapter 6 answers this question. The design can be implemented on two FPGAs to create a network diode featuring TCP. Furthermore, this project entails three subgoals to have the design established and to be able to validate its performance.

1. Design requirements
The design requirements stated in the first goal of this project should be achieved as requested from the original assignment provided by Technolution:

- **Minimise resource utilisation**
  The final design presented in Chapter 6, is developed by minimising the amount of hardware resources. The simulation tests performed in Chapter 4 were designed to uncover the parameters which could be reduced. Therefore, the complexity of the design should be reduced. Each additional instruction to the segments could introduce extra delay or require more computational power. The simulation results from Chapter 5 show the maximum used buffer space for certain configurations. The test with the minimal value is considered the best option for each run. Therefore, our design does not include a congestion avoidance algorithm and no TCP SACK option. The buffer at the red FPGA can be of size 15 KB, when the throughput is 100 Mbps.

- **Maintain a reliable connection**
  The connection is considered unreliable, when the diode is unable to transfer all the data from the original source to the original receiver. In our design this can only be the case when the amount of buffered data does not reach a maximum. By virtue of the TCP operations, all other errors are being handled.

- **Maximise data flow to 10 Gbps**
  The data rate for a FPGA is determined by the number of instructions which can be executed within a clock cycle and the clock frequency. Hence, the complexity of the design is minimised to prevent extra cycles are required for a segment to process. In Chapter 6, the data flow for both FPGAs is depicted

(Figures 6.5 and 6.6. The packets will arrive and be converted to a AXI-4 stream before processed. Every module will pass (a part of the segment) to the next module, based on the values read from the header fields. Processing 64 bits in a clock cycle with a clock frequecy of 156.25 MHz will feature a 10 Gbps data rate.

- **Manage multiple connections simultaneously**
  Our design does not include the option to distinguish different connections and manage them. However, our design does include room for this option to implement, which will be discussed in Section 7.3.

2. Develop a model for system evaluation
To validate the performance and to uncover the essential parameters for the design configuration, a model was required to determine the behaviour of the system. The model in Chapter 3 describes the model as two separate TCP streams, a black and a red stream. From this model the throughput of the system can be broken down to a few crucial parameters. The results is Equation (3.6), which can be used to evaluate the system on every aspect on TCP. To validate this equation and to evaluate the system, the simulation model presented in Chapter 4 will be used. A TCP Diode module in OMNeT++ can simulate the behaviour of the network diode in a realistic environment. This module allows to adjust the TCP configuration as well as the diode configuration as well. Therefore, this simulation module can be used for further research.

3. Design an accurate block diagram
To implement the configuration on a FPGA, we developed a high-level hardware design presented as a block diagram. The design is described in Chapter 6 and depicted in Appendix D. The design features a network diode consisting two FPGAs. One FPGA for each side of the diode. The design is implemented with the recommended configurations following the results from Chapter 5. Therefore, there is no congestion avoidance algorithm and no TCP SACK implementation. However, the design is modular, which allows to integrate new features.

Other contributions
During the project, we invested a considerable amount of time in the understanding of the TCP mechanisms and to simulate the network diode. Therefore, we developed scripts and programs to assist us, which are not included in this thesis. The following methods are used to contribute to achieving the project goals:

- A *python implementation of the TCP socket* was developed, which was able to process every of the TCP mechanisms. This socket was used to explore all possibilities that TCP provides. This allowed us to quickly test additional TCP options.
- For the selection of the simulation software, multiple simulators were tested. The three most promising simulators were used for further examination. For all three simulators (Mininet, NS-3, and OMNeT++), a complete network diode module was developed to test their capabilities.
- To interpret the output of OMNeT++, the data had to be converted to convenient values. Therefore, multiple scripts from Wireshark and Matlab were used to interpret the behaviour of the TCP streams.

7.3. Recommendations
The final result of the project is a proposed framework which should be used as the backbone of a VHDL project. The framework is presented by functional blocks and corresponding signals. those implementations could be easily extracted from the design parameters from our simulation model. The framework describes the functionality of the blocks and allows room for additions without a need of redesigning the whole project. However, the simulation model which was described in Chapter 4 allows a user to implement new or additional features for testing purposes. In this section, we will discuss the methodology performed during this project and we will propose additional features to be implemented in the future.

7.3.1. Discussion
To narrow down the scope of parameters to be investigated, we developed simplifications during this project. those simplifications neglect some rules or assumed some of the variables of the system. The simplifications allows us to focus on specific components of the system, which are highlighted in this thesis. We will discuss and elaborate the assumptions we used for our design.

- **Error rate**
  During the simulation tests, the error rate was not varied between different values. In our simulations, the error rate probability would lose one in 10.000 packets. The error rate is dependent on many other components or settings in the network. Therefore, it is very hard to predict or determine the probable error rate for a connection. An error could occur due to the network architecture. Each hop the packet has to travel is an extra risk on losing or corrupting the packet. The configuration on each passing device has an influence on the rate of lost packets. Even the priority setting for the connection is of strong influence.

- **Maximum segment size**
  In our simulation tests, the maximum segment size (MSS) is fixed to the same size on both devices. The MSS has the default value, with the assumption this is the least amount all devices should be able to cooperate with. If the size is not equal for the red and the black network, a new issue arises. The segments should then be fragmented, which will introduce extra complexity in the sequence calculation of the design. If the black network has a larger MSS than the red network, this will also introduce extra overhead at the red network segments. Extra overhead will result in a reduction of the maximum throughput.

- **Round-trip time**
  The round-trip time (RTT) in our simulation is set by a delay in the connection between two devices. This has two anomalies compared to a "real-world" system. First, the RTT is fixed for the complete simulation run. In a real-world system the RTT would vary over time and is not likely to be constant during the complete session. It would be a convenient feature to implement a time-varying RTT in the simulator. Secondly, the RTT is determined as twice the amount of the connection delay. The physical link is likely to be realised by an asynchronous pathway.

- **Data rate**
  The framework of our design should be able to achieve a data rate of 10 Gbps. However, our simulation could not test an endurance run with this speed. The TCP module implementation in the INET framework of OMNet++ is limited by the sequence number. A TCP segment has a 32-bit sequence number. The INET implementation does not restart the counting if the maximum number is reached. This allows the simulation to only run for 3.4 seconds.

- **Re-ordering**
  Our design has no buffer implemented on the black FPGA. Therefore, that side is unable to re-order packets. If a packet arrives with a larger sequence number than the module is expecting ($SEG.SEQ > RCV.NXT$), than the module is not able to store the segment temporarily and wait for the missing segment. Therefore, if a packet is received out-of-order, the module is not able to acknowledge the packet and will not send it over the diode. This will result in a "detection" of a packet loss and does require a recovery of the connection. This will slow down the black TCP stream and unloads the red TCP stream to empty the buffer.

7.3.2. Future work

The design of the network diode features the basic mechanisms of the TCP functionality. The diode has the capacity to establish a TCP data stream between two separated domains, ensuring unidirectional data flow. The framework proposed in Chapter 6 allows to implement this functionality on a piece of hardware realised by two FPGAs. The diode however, is not like the intelligent diodes as discussed in Chapter 2. All processes and operations of the diode are being executed on hardware. Therefore, the diode is not re-configurable and once installed, no extra features can be added to the design. To improve the performance or append new features to the diode, the following options should be considered:

- **Congestion algorithm**
  Our design does not implement a congestion avoidance algorithm at the red FPGA. The simulation results showed only an high increase in buffered data while a congestion algorithm was applied. The red part of the network is connected to a trusted domain and in full control of the user. The network administrator could configure the network to prioritise the stream coming from the diode, which should

negate the necessity of a CA algorithm. The CA algorithms provided in the INET-framework are not up-to date. The common TCP cubic algorithm is not implemented. Although, a convenient implementation in OMnet++ for this is provided by Singh [61].The simulation did not run test scenarios with other traffic. Allowing other traffic during the diode stream is likely to require a CA algorithm to maintain a stable throughput.

- **TCP timestamp**
  During high-speed data transfers, many segments are coming by in a short period. The limited amount of available sequence numbers could arise an issue with ambiguity of equal sequences. A solution for this is proposed by RFC 1323 [32]. This TCP options does add a timestamp value to the segment. By this method, both devices are able to identify which sequence number was first and resolves the ambiguity issue. The OMnet++ framework does include the timestamp option. However, OMnet++ is not able to cylce through the sequence numbers. If the last sequence number is reached it does not restart from the beginning, but rather end the connection. Therefore, it was unable to test the influence of this TCP option in our design.

- **RTT estimation**
  Several algorithms include the calculation of the round trip time. Implementing a RTT estimation on the black FPGA, does allow for extra control of the throughput. The window size is fixed, and by knowing the current RTT the black FPGA is able to estimate the load on the red network. The black FPGA could limit the throughput by denying packets from the black source. The black part of the diode should then be in more control of unloading the red network.

- **Multiple connections**
  The current implementation does allow only one connection over the diode. Although, there is the possibility to implement multiple connections. A networking interface should be managed by the soft-core. The incoming packets should then be multiplexed and tagged to notify the red part of the diode to what connection the packet belongs. The connection capacity should be managed by the diode and should distribute balanced resources. The connections could be prioritised to give a larger share of the resources to particular streams. This is done by throughput limitation at the black FPGA. The red FPGA should segment parts of the buffer for the corresponding packets.

- **Queuing**
  Congestion avoidance algorithms are not the only management systems to prevent a congestion on the network. Another popular procedure, is the queueing method. The queue represents the segments in the buffer. With a simple drop-tail queue, a segment will only be added to the queue if there is space left. An active queueu management can even predict a full queue and will drop packets on purpose to prevent a congestion. However, the black FPGA is not aware of the saturation of the buffer at any moment. But the black FPGA could predict the saturation of the buffer with the assumption of a fixed throughput. If the buffer is approaching its full capacity considering the prediction, the black FPGA is allowed to drop packets on purpose and unload the red FPGA eventually.

- **Diode protocol**
  The protocol used for the electrical connection between the two FPGAs (the diode connection), is relatively simple. If a segment arrives in order at the black FPGA it will be sent over the diode. The black FPGA does simultaneously transmit an acknowledgement the to black source and transmit the segment over the diode. This protocol could be improved by some simple adjustments. First of all, the segment size is equal for both FPGAs, which does allow them to keep the same sequence number, because they do not need to be fragmented. The sequence number is attached to the segment and stored in the buffer on the red side. This is an overhead which could be reduced. The black FPGA is not able to re-order segments and therefore, it will transmit the segments sequentially over the diode. The sequence number is unnecessary to store in the buffer if all segments arrive in-order. To improve the throughput, the black FPGA could transmit an out-of order segment over the diode and not acknowledge it. The red FPGA will then already be able to send this segment, while the black connection is still recovering. To implement this, the black FPGA should have an register that keeps track to what segments are already sent over the diode. Otherwise it will send duplicates to the buffer. In this case, the sequence number should be attached or another method should be used to identify the order of the segments.

A

Selective Acknowledgement

The TCP NewReno's fast recovery algorithm gets issues when more than one consecutive packets are lost. The acknowledgement method of TCP handles acknowledgements cumulative. This property limits the capability to detect no more than one packet loss during one RTT. For example, when a second or a third packet get lost after the first lost packet, the receiver would reply with a duplicate acknowledgement of the last received packet. This process is illustrated in Figure A.1. Furthermore, only one packets is retransmitted after the loss detection, Therefore, the recovery time for the fast recovery method of TCP NewReno is linear proportional with the number of packets lost. The recovery time could be reduced when the sender retransmits more than only one segment. Unfortunately the receiver can only notify one lost segment and not indicate any more. Retransmitting extra packets is unlikely to send the lost packets and increases the risk of congesting the network unnecessarily.

If the receiver could provide additional information about packet losses, the recovery time could be reduced. The sender is then allowed to retransmit several segments. The receiver should be able to notify the sender in one single feedback message, what packets got lost. The TCP NewReno's implementation of the Fast Recovery algorithm restricts the congestion window to be reduced only once per RTT period. This ensures the detection of a lost packet is in the worst case an interval of exactly one RTT. All losses will be notified to the sender within one RTT period. This process is depicted in Figure A.2

Mathis et al. proposed a solution to the limited information that cumulative acknowledgements could provide [46]. They developed the **Selective Acknowledgement** (SACK) option, to extend the TCP protocol. The SACK option allows the receiver to mark blocks of correctly received packets. The gaps in between the defined blocks are the lost segments. To use the SACK option, both devices must agree during negotiation, they are capable and willing to use this option. Therefore, SACK has two option kinds: one for negotiate the use of SACK, and the other for reporting the SACK-blocks. This does require two different kind numbers, because they have a different length and purpose. The maximum number of lost packets that can be reported by the SACK option is limited by the maximum size of the TCP header. The maximum size of the option field

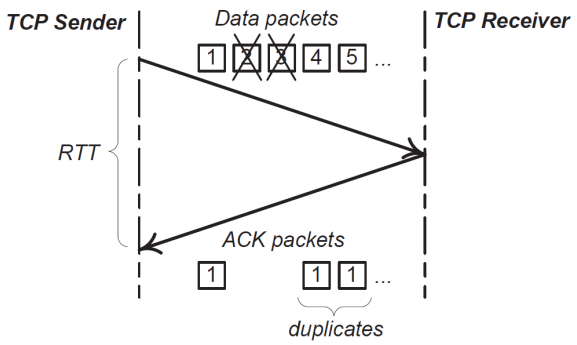


Figure A.1: Duplicate ACKs allow loss detection no sooner than after one RTT ([2] fig.19)

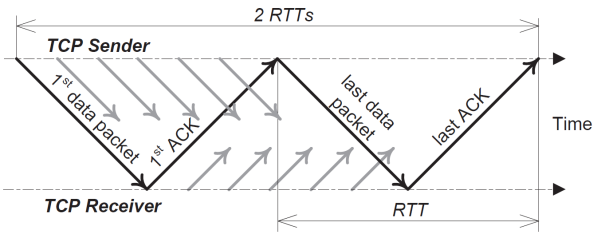


Figure A.2: The interval between the first and the lat data packets before reception of any acknowledgement is two times the RTT period. In the worst case, any lost packet is detectd after exactly one RTT ([2] fig.20)

is 40 bytes, which allows the SACK option to report up to 4 blocks of received data packets (1 byte for option kind, 1 byte for option length and four blocks of 8 bytes, containing 2 sequence numbers). The concept of the SACK blocks is depicted in Figure A.3

Although SACK provides extra information about out-of-order segments, it does not define any instructions for the congestion window. The rule of reducing the congestion window only once during one RTT period, does already allow the sender to detect losses within one periode. The SACK option itself only prevents unnecessary retransmits by reporting additional information about the lost packets. Mathis and Mahdavi proposed a congestion control algorithm using the additional information retreived from the SACK blocks. They define a recovery procedure called **Forward Acknowledgements** (FACK). The FACK algorithm provides a method to timely retransmit data. A lost packets is at least reported for one RTT period. Therefore, FACK requires the time of the last retrasnmission in order to detect a loss. FACK does not define a congestion window, instead it calculates the number of packets in transit. The algorithm keeps track of three state variables: 1) H, the highest sequence number of all sent data; 2) F, the forward-most sequence number of all acknowledged data (no sequence number higher than F, have been acknowledged); 3)R, the number of retransmitted data. The number of outstanding packets would then be: $H - F + R$ as is depicted in Figure A.4. This calculation can be utilised by the sender to decide if a new segment is allowed to be transmitted. If the number of outstanding segments is smaller than the current congestion window, the receiver should be able to accept more data segments.

The SACK option provides additional information to improve the recovery of a TCP connection. FACK features an extra mechanism to make more use of the provided information of SACK. Implementing SACK solely does improve the reliability of the TCP connection. In our case, SACK could even be more effective.

Implementing SACK in the data diode should provide the following advantages:

1. **Reduce buffer space**
   The buffer contains all the segments received from the black buffer, which are not yet sent to, or acknowledged by, the red receiver. The segments are removed from the buffer when they are acknowledged by the red receiver. The SACK option provides an additional method to acknowledge segments. The information provided in the SACK-blocks, can already be removed from the buffer, which creates extra space. Therefore, less buffer space is required.

2. **Improve retransmit pattern**
   The sender is better informed about the lost segments. The gaps between the SACK blocks report the lost segments in between correctly transmitted segments. The sender does not have to retransmit a bunch of segments which are not lost. Less segments will be retransmitted, which increases the efficiency of the stream. Furthermore, the total transmitted segments (including retransmits) will be reduced, which reduces potential congestion. The low risk on congestion, will increase the throughput of the connection.

3. **Reduce recovery time**
   The recovery time is reduces by using SACK and FACK. After recovery, the congestion control algorithms will increase the window. When the connection is recovered within a smaller period, the congestion window will be more quickly restored to the maximum value. Therefore, the throughput is restored in faster fashion.

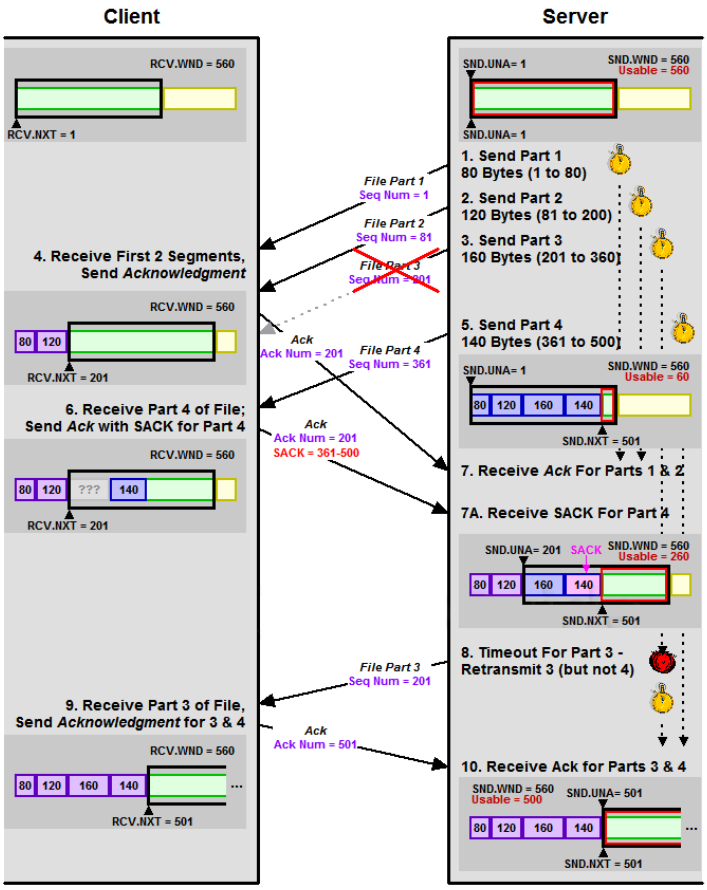


Figure A.3: Graphical representation of the SACK-option ([40] fig. 49-3)

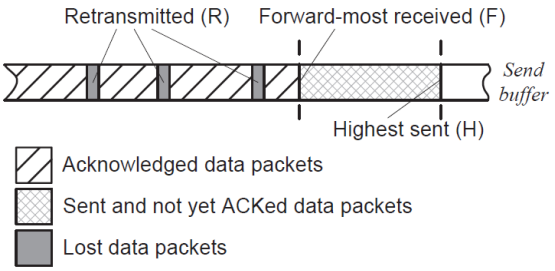


Figure A.4: $H - F + R$ is equal to the packets in transit ([2] fig. 21)

B

MiniNet scripts

MiniNet was tested to verify whether it was a suitable simulation solution for our system. MiniNet allows to create a complete network virtually. All hardware and connections will run as a "virtual machine" on the Mininet host device. Therefore, the simple module (Figure 3.3) has been implemented in a topology for MiniNet, which is listed below. The topology contains four hosts. The data diode is simulated by two hosts. One host in the black network and one host in the red network. The black source is connected to the data diode via a switch. At the red side, a switch connects the sender simulator and the red receiver. The diode is formed by the border between the diode hosts. Finally, each host will start a small virtual machine. The red diode host will run a python script that executes all the required steps for a receiving TCP socket. The black host will run a python script that executes all the required steps for the sending TCP socket. The global functions implemented in these sockets are presented in a tree graph in Figures B.1 and B.3. The source code for the implemented Python sockets can be requested from my Github repository [38].

Listing B.1: topo-datadiode.py

```
1  """Custom topology for simulation of network diode
2
3  The black and the red network are connected by the simulator hosts:
4
5
6              ----------BLACK----------           |           ----------RED----------
7    Black_source(host) -- Switch -- Receiver_simulator(host) --- Source_simulator(host) --Switch-- Red_receiver(host)
8
9  """
10
11 from mininet.topo import Topo
12
13 class MyTopo( Topo ):
14     "Simple topology example."
15
16     def build( self ):
17         "Create custom topo."
18
19         # Add hosts and switches
20         Server = self.addHost( 'h1' )
21         Switch_black = self.addSwitch('s1')
22         Client_simulator = self.addHost( 'h2' )
23         Server_simulator = self.addHost( 'h3' )
24         Switch_red = self.addSwitch('s2')
25         Client = self.addHost( 'h4' )
26
27         # Add links
28         self.addLink( Server, Switch_black )
29         self.addLink( Switch_black, Client_simulator)
30         self.addLink( Client_simulator, Server_simulator )
31         self.addLink( Server_simulator, Switch_red)
32         self.addLink( Switch_red, Client )
33
34
35 topos = { 'mytopo': ( lambda: MyTopo() ) }
```

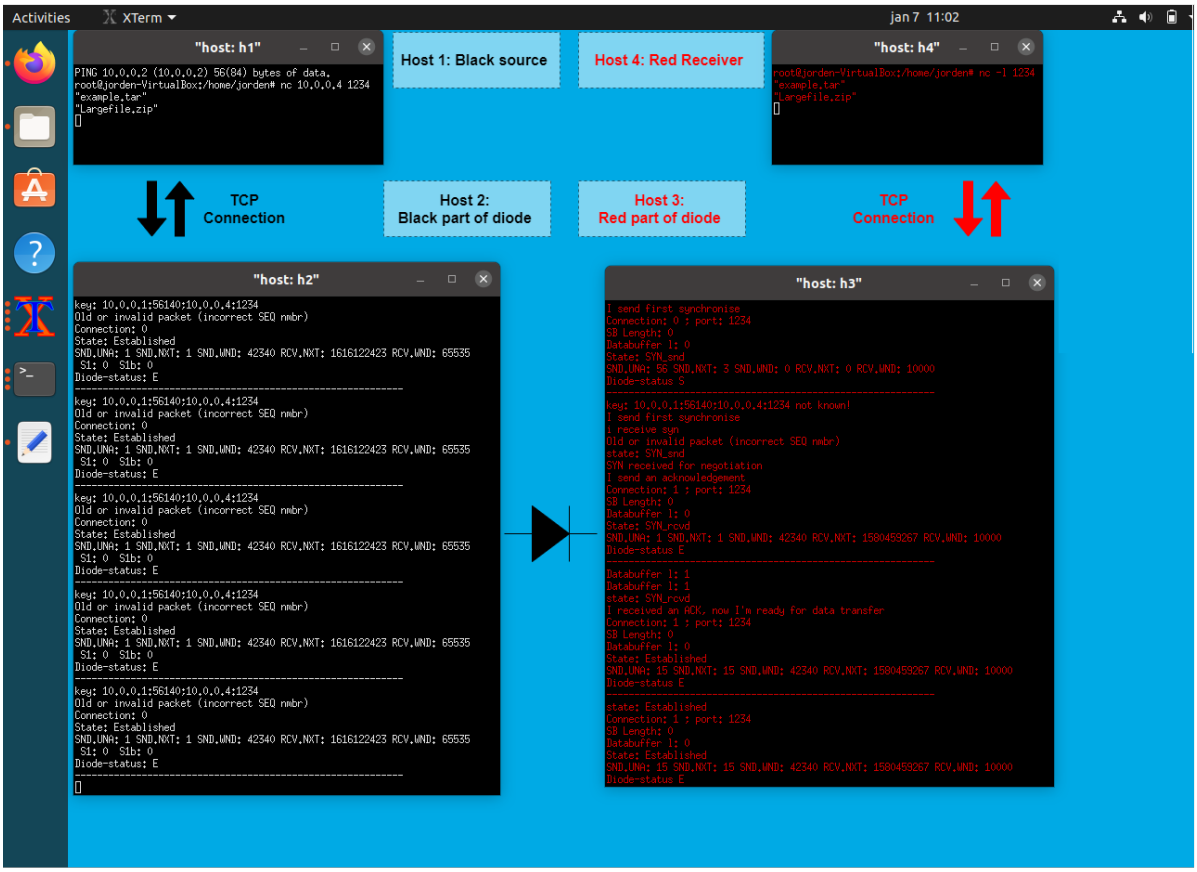


Figure B.2: Example of the Mininet set-up running in linux Ubuntu

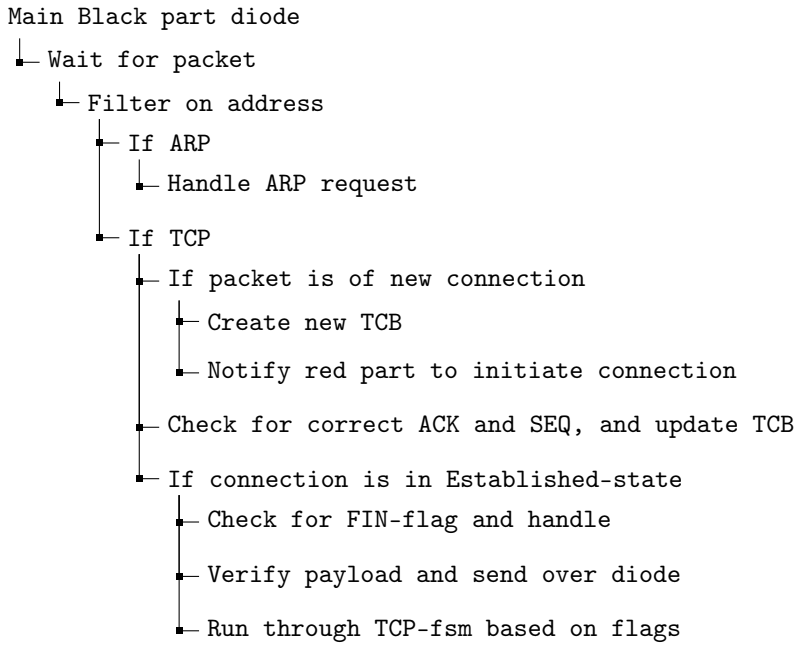


Figure B.1: Functionality tree of the black data diode host

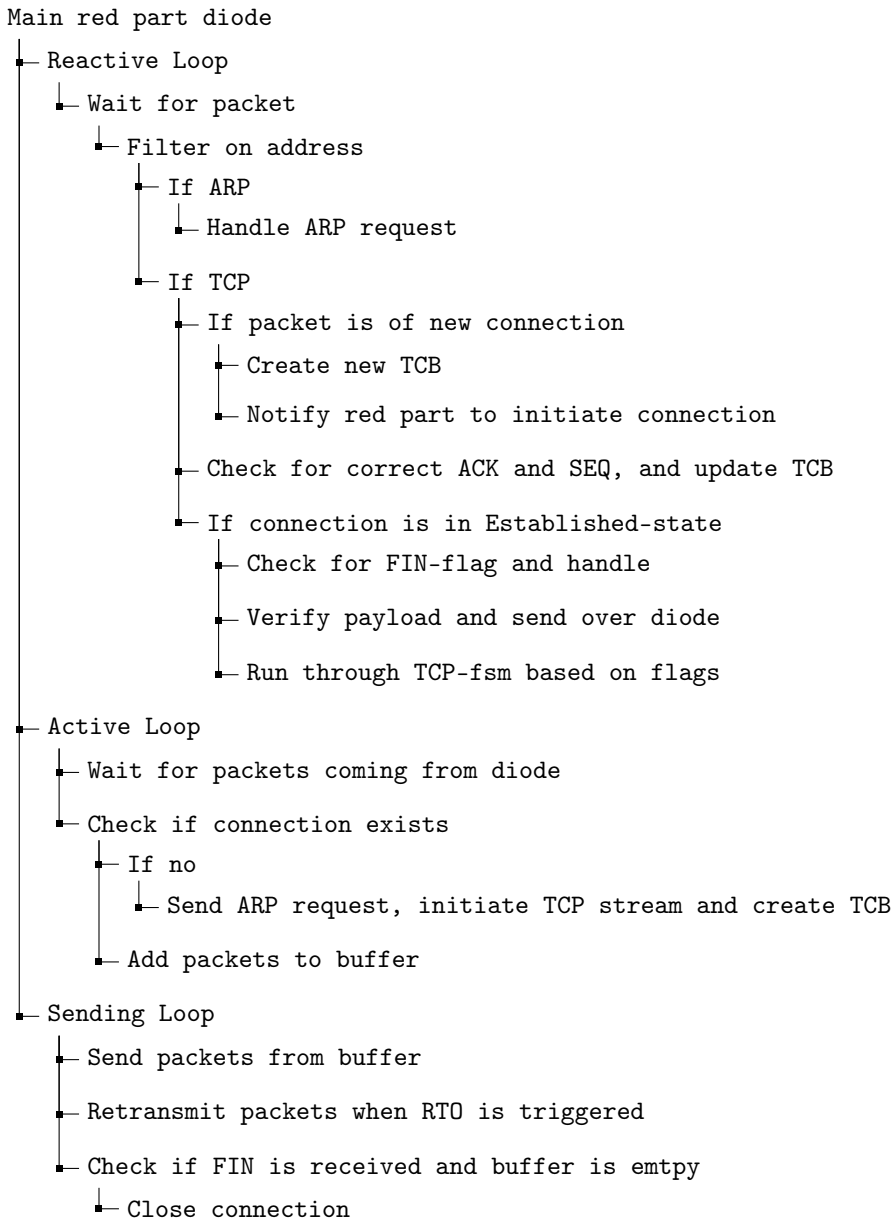


Figure B.3: Functionality tree of the black data diode host

C

OMNeT++ scripts

The testing module for the network diode is depicted in Figure C.1. All three the devices are a standardhost, featuring all possibilities to implement network features. The black source will create an "unlimited" TCP stream to send to the red receiver over the diode. The data diode node will transfer the payload from the black to the red source. The data diode has two Ethernet connections implemented; one connection for a TCP socket in each network. The black TCP socket will receive the packets, send acknowledgements and pass the payload to the red socket. The red socket is responsible for transferring the packets to the "original" red receiver. OMNeT++ provides a IDE that is able to fluently switch between source code and graphical design. Therefore, the source code of the module design is listed below. In this listing you will find the parameters used in the simulation. The module itself describes the "hardware" of the devices on a network level. The other parameters are defined in the sub-modules like the application. The black source will run an application to actually transmit data to the data diode. Likewise does the red receiver run an application that takes care of incoming segments. The data diode application will execute the instructions of both red and black TCP sockets. The parameters for the simulation are defined in the configuration file. The source code for the application, including the configuration files, can be requested from my private Github repository [39].

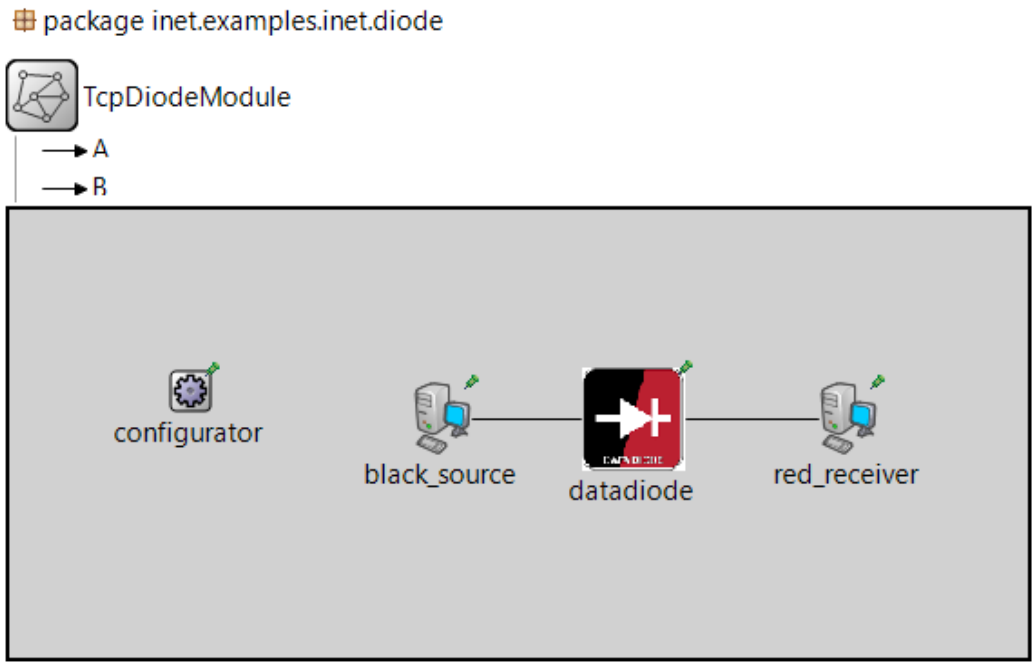


Figure C.1: Graphical overview of the TCPDiodeModule design in OMNeT++

Listing C.1: TcpDiodeModule.ned

```
1  //
2  // Copyright (C) 2000 Institut fuer Telematik, Universitaet Karlsruhe
3  //
4  // This program is free software; you can redistribute it and/or
5  // modify it under the terms of the GNU General Public License
6  // as published by the Free Software Foundation; either version 2
7  // of the License, or (at your option) any later version.
8  //
9  // This program is distributed in the hope that it will be useful,
10 // but WITHOUT ANY WARRANTY; without even the implied warranty of
11 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
12 // GNU General Public License for more details.
13 //
14 // You should have received a copy of the GNU General Public License
15 // along with this program; if not, write to the Free Software
16 // Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
17 //
18
19 package inet.examples.inet.diode;
20
21 import inet.common.misc.NetAnimTrace;
22 import inet.networklayer.configurator.ipv4.Ipv4NetworkConfigurator;
23 //import inet.node.inet.CopyStandardHost;
24 import inet.node.inet.StandardHost;
25 import ned.DatarateChannel;
26 import inet.node.base.DiodeNodeBase;
27
28
29 network TcpDiodeModule
30 {
31     parameters:
32         double datarateA @unit(bps) = default(1Gbps);
33         double datarateB @unit(bps) = default(1Gbps);
34
35         double delayA @unit(s) = default(5ms);
36         double delayB @unit(s) = default(5ms);
37
38         double perA = default(0); //Probability Packet Error Rate
39         double perB = default(0);
40
41     types:
42         channel A extends DatarateChannel
43         {
44             datarate = datarateA;
45             delay = delayA;
46             per = perA;
47         }
48         channel B extends DatarateChannel
49         {
50             datarate = datarateB;
51             delay = delayB;
52             per = perB;
53         }
54     submodules:
55         black_source: StandardHost {
56             parameters:
57                 @display("p=239,115;i=device/pc3");
58         }
59         //DiodeNodeBase or StandardHost
60         datadiode: StandardHost {
61             parameters:
62                 @display("p=344,115;i=jorden/datadiode");
63         }
64         red_receiver: StandardHost {
65             @display("p=463,115;i=device/pc3");
66         }
67         configurator: Ipv4NetworkConfigurator {
68             parameters:
69                 @display("p=100,100;is=s");
70         }
71     connections:
72         black_source.ethg++ <--> A <--> datadiode.ethg++;
73         datadiode.ethg++ <--> B <--> red_receiver.ethg++;
74 }
```

D

Block diagram schematics

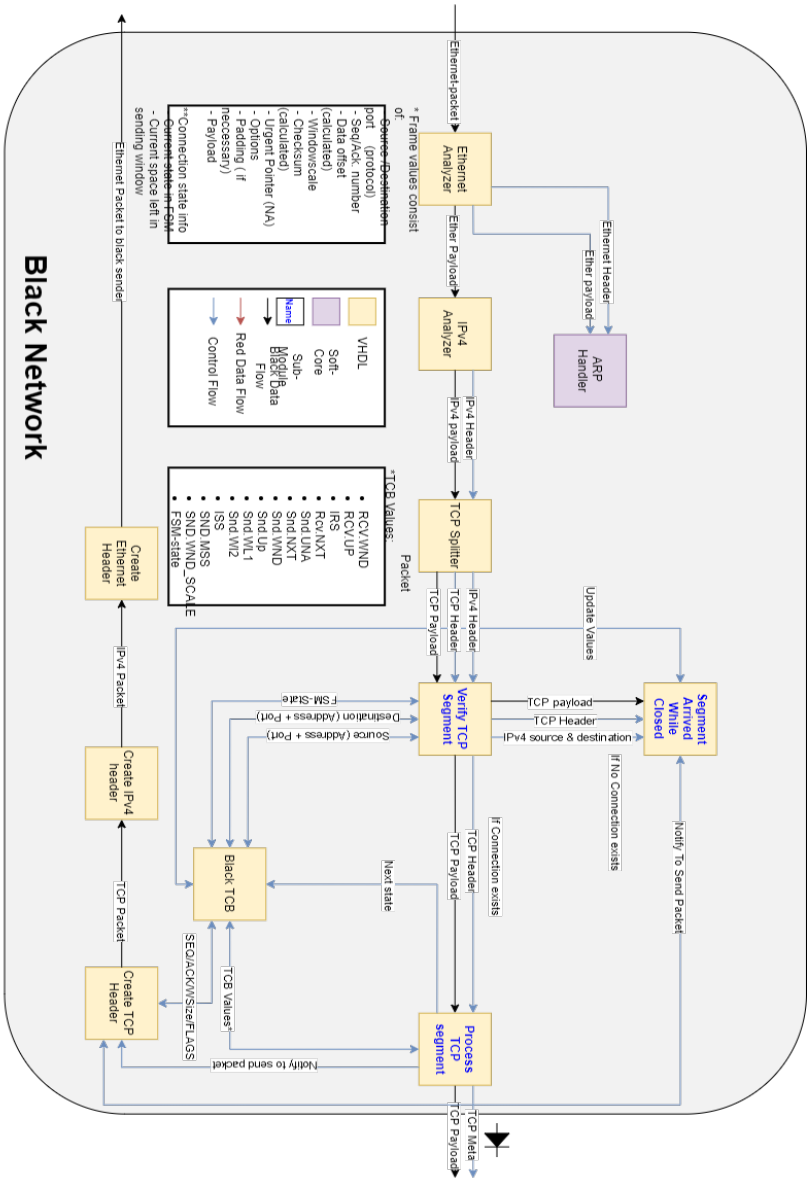


Figure D.1: Top level overview of the functional block diagram of black part of the network diode

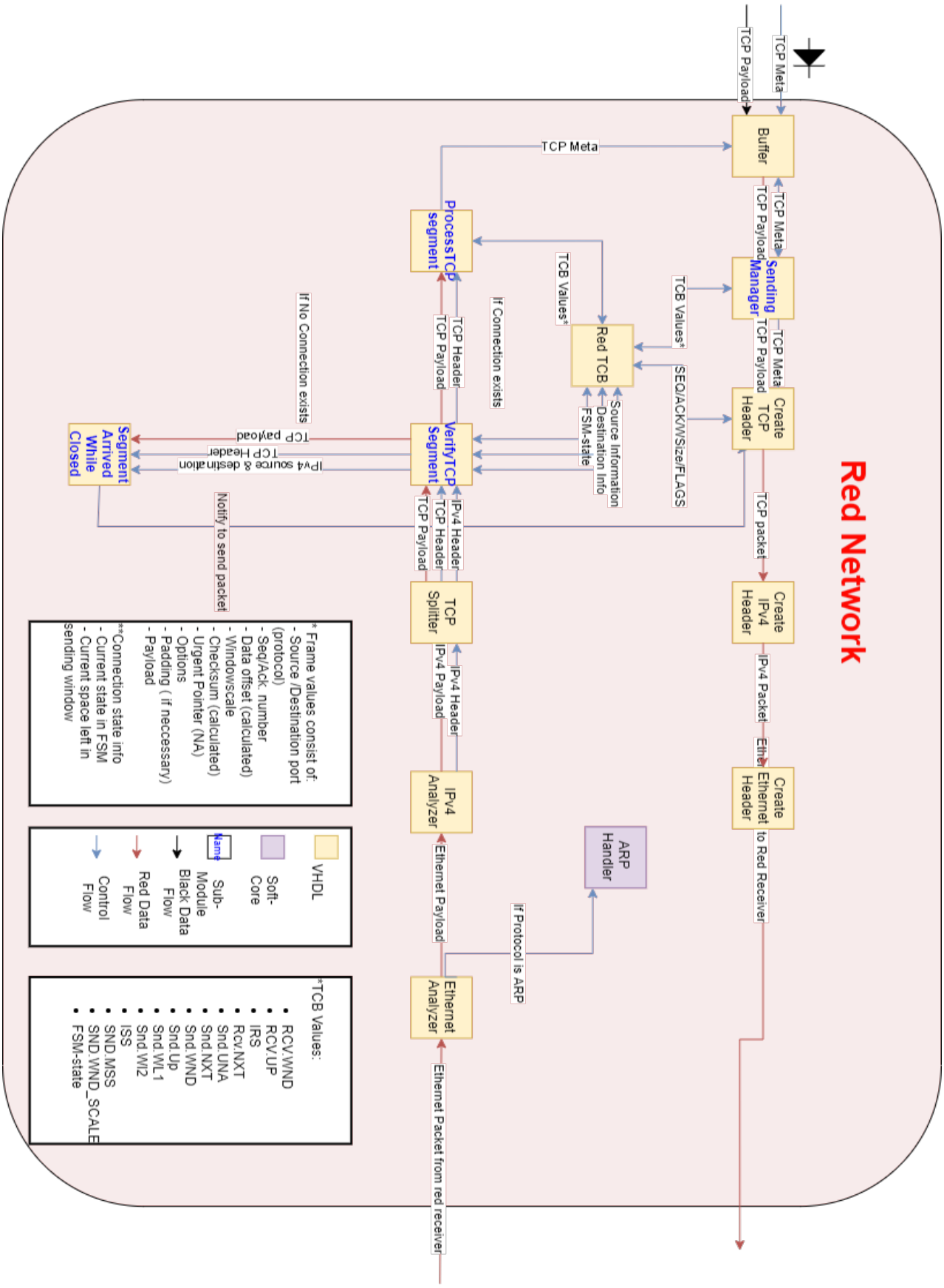


Figure D.2: Top level overview of the functional block diagram of red part of the network diode

Bibliography

[1] Advenica. Data diodes, n.d. URL https://www.advenica.com/en/cds/data-diodes. Accessed: 20-10-2020.

[2] Alexander Afanasyev, Neil Tilley, Peter Reiher, and Leonard Kleinrock. Host-to-host congestion control for tcp. *IEEE Communications surveys & tutorials*, 12(3):304–342, 2010.

[3] Mark Allman, Vern Paxson, and William Stevens. Rfc2581: Tcp congestion control, 1999.

[4] Eitan Altman, Konstantin Avrachenkov, Chadi Barakat, Arzad Alam Kherani, and BJ Prabhu. Analysis of mimd congestion control algorithm for high speed networks. *Computer Networks*, 48(6):972–989, 2005.

[5] ARM AMBA. Axi4-stream protocol vl. 0, 4.

[6] Guido Appenzeller, Isaac Keslassy, and Nick McKeown. Sizing router buffers. *ACM SIGCOMM Computer Communication Review*, 34(4):281–292, 2004.

[7] Arbit. Arbit data diode & arbit trust gateway: Cross domain solutions: Cyber security, Mar 2020. URL https://arbitcds.com/products/. Accessed: 20-10-2020.

[8] Guy AvrahamGuy. How to check the tcp congestion control algorithm flavour in ubuntu, Jan 2018. URL https://superuser.com/questions/992919/how-to-check-the-tcp-congestion-control-algorithm-flavour-in-ubuntu.

[9] Praveen Balasubramanian and Microsoft. Updates on windows tcp, n.d.

[10] Wei Bao, Vincent WS Wong, and Victor CM Leung. A model for steady state throughput of tcp cubic. In *2010 IEEE Global Telecommunications Conference GLOBECOM 2010*, pages 1–6. IEEE, 2010.

[11] Wolfgang Bokämper. Industrie 4.0 security guidelines: Recommendations for actions, 2016.

[12] Robert Braden. Rfc1122: Requirements for internet hosts-communication layers, 1989.

[13] Dah-Ming Chiu and Raj Jain. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN Systems*, 17(1):1 – 14, 1989. ISSN 0169-7552. doi: https://doi.org/10.1016/0169-7552(89)90019-6. URL http://www.sciencedirect.com/science/article/pii/0169755289900196.

[14] Scott W. Coleman. *The Definitive Guide to Data Diode Technologies*. Owl Cyber Security, 2019.

[15] Douglas E Comer. *Internetworking with tcp/ip*. Addison-Wesley Professional, 2013.

[16] PA Consulting. Data diode, n.d. URL https://www.paconsulting.com/services/product-design-and-engineering/data-diode/. Accessed: 20-10-2020.

[17] controlledinterfaces. Optical data diode solutions, n.d. URL https://www.controlledinterfaces.com/. Accessed: 20-10-2020.

[18] Casey Crane. 42 cyber attack statistics by year: A look at the last decade, Jul 2020. URL https://sectigostore.com/blog/42-cyber-attack-statistics-by-year-a-look-at-the-last-decade/.

[19] ZNO DATA-DIODE, n.d. URL https://www.filbico.pl/ZNO-EN.html. Accessed: 20-10-2020.

[20] DataFlowX. Next generation data diode solution, n.d. URL https://www.dataflowx.com/. Accessed: 20-10-2020.

[21] Gary Dickson and Alan Lloyd. *Open systems interconnection*. Prentice-Hall, 1992.

[22] donaldh. Which congestion control algorithm is used by the tcp stack in os x?, Jul 2015. URL https://superuser.com/questions/865896/which-congestion-control-algorithm-is-used-by-the-tcp-stack-in-os-x.

[23] Sally Floyd and Kevin Fall. Promoting the use of end-to-end congestion control in the internet. *IEEE/ACM Transactions on networking*, 7(4):458–472, 1999.

[24] Sally Floyd, Tom Henderson, and Andrei Gurtov. Rfc3782: The newreno modification to tcp's fast recovery algorithm, 2004.

[25] ATM Forum. *ATM user-network interface (UNI) specification version 3.1*. Prentice Hall, 1995.

[26] Fox-IT. Beveiligingswaarde fox crypto datadiode bestempelt als 'zeer geheim' – fox-it, november 2019. URL https://www.fox-it.com/nl/actueel/persberichten/beveiligingswaarde-fox-crypto-datadiode-bestempelt-als-zeer-geheim/.

[27] Mario Gerla and Leonard Kleinrock. Flow control: A comparative survey. *IEEE Transactions on Communications*, 28(4):553–574, 1980.

[28] James Graham, Jeffrey Hieb, and John Naber. Improving cybersecurity for industrial control systems. In *2016 IEEE 25th International Symposium on Industrial Electronics (ISIE)*, pages 618–623. IEEE, 2016.

[29] IANA. Transmission control protocol (tcp) parameters, Apr 2020. URL https://www.iana.org/assignments/tcp-parameters/tcp-parameters.xhtml#tcp-parameters-1.

[30] infodas. Sdot diode, Jun 2020. URL https://www.infodas.de/en/products/sdot_cross_domain_solutions/data_diode/. Accessed: 20-10-2020.

[31] BAE Systems | International. Data diode solution™, n.d. URL https://www.baesystems.com/en/product/data-diode-solution. Accessed: 20-10-2020.

[32] V. Jacobsen. TCP Extensions for High Performance. RFC 1323, RFC Editor, May 1992. URL https://www.rfc-editor.org/rfc/rfc1654.txt.

[33] Van Jacobson. Interpacket arrival variance and mean. *Letter to the TCP-IP mailing list*, 15, 1987.

[34] Van Jacobson. Congestion avoidance and control. In *ACM SIGCOMM computer communication review*, volume 18, pages 314–329. ACM, 1988.

[35] Phil Karn and Craig Partridge. Improving round-trip time estimates in reliable transport protocols. *ACM SIGCOMM Computer Communication Review*, 25(1):66–74, 1995.

[36] Karamjeet Kaur, Japinder Singh, and Navtej Singh Ghumman. Mininet as software defined networking testing platform. In *International Conference on Communication, Computing & Systems (ICCCS)*, pages 139–42, 2014.

[37] Tom Kelly. Scalable tcp: Improving performance in highspeed wide area networks. *ACM SIGCOMM computer communication Review*, 33(2):83–91, 2003.

[38] J. Kerkhof. Tcp socket mininet, 2020. URL https://github.com/KerkhofJ/TCP-socket.

[39] J. Kerkhof. Tcp diode project, 2020. URL https://github.com/KerkhofJ/TCP_Diode_Project.

[40] Charles M. Kozierok. *The TCP/IP-Guide: a comprehensive, illustrated Internet protocols reference*. No Starch Press, 2009.

[41] J. Kurose and K. Ross. *Computer Networking: A Top-Down Approach, Global Edition*. Pearson Education Limited, 2017. ISBN 9781292153605. URL https://books.google.nl/books?id=IUh1DQAAQBAJ.

[42] D. Lee, B. E. Carpenter, and N. Brownlee. Observations of udp to tcp ratio and port numbers. In *2010 Fifth International Conference on Internet Monitoring and Protection*, pages 99–104, 2010.

[43] Peter Loshin. *TCP/IP clearly explained*. Morgan Kaufmann Publishers, 2003.

[44] et al. M. Allman. Tcp congestion control. RFC 5681, RFC Editor, 09 2009. URL https://tools.ietf.org/html/rfc5681.

[45] Saverio Mascolo, Claudio Casetti, Mario Gerla, Medy Y Sanadidi, and Ren Wang. Tcp westwood: Bandwidth estimation for enhanced transport over wireless links. In *Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 287–297, 2001.

[46] M Mathis, J Mahdavi, S Floyd, and A Romanov. Rfc2018 tcp selective acknowledgment options. internet engineering task force, 1996.

[47] Matthew Mathis, Jeffrey Semke, Jamshid Mahdavi, and Teunis Ott. The macroscopic behavior of the tcp congestion avoidance algorithm. *ACM SIGCOMM Computer Communication Review*, 27(3):67–82, 1997.

[48] Krishanu Maulik and Bert Zwart. An extension of the square root law of tcp. *Annals of Operations Research*, 170(1):217–232, 2009.

[49] Inc. Microchip Technology, 2020. URL https://microchipdeveloper.com/ethernet:overview.

[50] Piet Mieghem. *Data Communications Networking*. Techne Press, 2006.

[51] Jeffrey C Mogul and Christopher A Kantarjiev. Retrospective on" fragmentation considered harmful". *ACM SIGCOMM Computer Communication Review*, 49(5):41–43, 2019.

[52] John Nagle. Rfc0896: Congestion control in ip/tcp internetworks, 1984.

[53] Teunis J Ott. Transport protocols in the tcp paradigm and their performance. *Telecommunication Systems*, 30(4):351, 2005.

[54] V Paxson, M Allman, and Computing TCP's Retransmission Timer. Rfc 2988. *Computing TCP's retransmission Timer*, 2000.

[55] Jon Postel. Rfc793—transmission control protocol, darpa internet program, protocol specification. *Information Sciences Institute/Defense Advanced Research Projects Agency. Retrieved November*, 19:2009, 1981.

[56] RolloosSystems. Datadiode: Prevent your rig from being hacked, Apr 2018. URL https://www.rolloos.com/en/solutions/cyber-security/datadiode/. Accessed: 20-10-2020.

[57] Mats Holm Rosbach. Verification of network simulators: The good, the bad and the ugly. Master's thesis, University of Oslo, 2012.

[58] Rovenma. Kindi data diode, n.d. URL https://www.rovenma.com/kindi-data-diode-devices/. Accessed: 20-10-2020.

[59] Deep Secure. Data diode, august 2016. URL https://www.deep-secure.com/data-diode.php. Accessed: 20-10-2020.

[60] | Waterfall Security. Unidirectional security gateways: Waterfall security, May 2020. URL https://waterfall-security.com/unidirectional-security-gateways/. Accessed: 20-10-2020.

[61] Navdeep Singh and Rajesh Kumar. Tcp cubic implementation proposal using omnet+. *Proceedings of 4th International Conference on Advancements in Engineering & Technology (ICAET-2016)*, 2016.

[62] W. Richard. Stevens. *TCP/IP illustrated, volume 1: the protocols*. Addison-Wesley Pub., 1994.

[63] US ICS Cyber Emergency Response Team. Recommended practice: Improving industrial control systems cybersecurity with defense-in-depth strategies. *Department of Homeland Security, Washington, DC, USA, www. ics-cert. us-cert. gov/sites/default/files/recommended _practices/NCCIC_ICS-CERT_Defense_in_Depth_2016_S508C. pdf*, 2016.

[64] Technolution. Primediode 3010 data diode, Oct 2020. URL https://www.technolution.com/prime/nl/primediode-3010-data-diode/.

[65] Belma Turkovic, Fernando A Kuipers, and Steve Uhlig. Fifty shades of congestion control: A performance and interactions evaluation. *arXiv preprint arXiv:1903.03852*, 2019.

[66] vadosecurity. Vado security data didoe solution: Hardware & full agents vm support, n.d. URL `https://www.vadosecurity.com/`. Accessed: 20-10-2020.

[67] Siemens Mobility Global Website. Firewalls are outdated - how to make your network secure with data diodes, september 2019. URL `https://www.mobility.siemens.com/global/en/portfolio/rail/automation/reports/data-capture-unit.html`. Accessed: 20-10-2020.

[68] Klaus Wehrle, Mesut Günes, and James Gross. *Modeling and tools for network simulation*. Springer Science & Business Media, 2010.

[69] Jörg Widmer, Robert Denda, and Martin Mauve. A survey on tcp-friendly congestion control. *IEEE network*, 15(3):28–37, 2001.

[70] Minicy Catom Software Engineering Ltd. www.catom.com. Vit-400 cyber network security, 2017. URL `http://www.wizlan.com/WizLan/Templates/showpage.asp?DBID=1&TMID=108&FID=294&PID=874&IID=881`. Accessed: 20-10-2020.