Øyvind Aasen

# Using Bi-directional Data Diodes to Limit Propagation of Network Attacks

Master's thesis in Information Security
Supervisor: Prof. Slobodan Petrovic
July 2019

Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Information Security and Communication Technology

NTNU
Norwegian University of
Science and Technology

Norwegian University of
Science and Technology

# Using Bi-directional Data Diodes to Limit Propagation of Network Attacks

Øyvind Aasen

01-06-2019

Master's Thesis
Master of Science in Information Security
30 ECTS
Department of Information Security and Communication Technology
Norwegian University of Science and Technology,

Supervisor: Prof. Slobodan Petrovic

# Preface

This Master's thesis at the Department of Information Security and Communication Technology at NTNU Gjøvik was carried out during the spring semester of 2019.

We assume that the reader of the thesis is knowledgeable about computer science with an interest in cyber security and networking.

01-06-2019

# Acknowledgment

I would like to thank my family and friends for their support during the work on this thesis. A special huge thanks to my father for helping me improve the writing and fixing my 'many' spelling mistakes and grammar faults in the thesis. Any remaining errors are my own. I would also like to thank my Supervisor Slobodan Petrovic for helping me with the thesis by helping me figure out what I should focus on when during my work as well as pushing me in a more scientific direction when I often moved towards a more technical direction. And for the help with finding a better title for the thesis in addition to providing feedback on what I have written and what I should improve in this report as well as grammar and spell checking.

Ø.Aa.

# Abstract


Most networks are vulnerable to many kinds of attacks on different devices. When an attacker gains access to one of the devices in a network he can then use it to attack other devices in the network. WannaCry and NotPetya are well known attacks that caused much damage. Compartmentalisation of the network is often used to minimise the number of reachable devices that are vulnerable against such attacks.

In this thesis a novel use of data diodes called a bi-directional data diode is introduced. A bi-directional data diode replaces one traditional bi-directional Ethernet link with two data diodes. The data diodes are connected in opposite directions. This configuration provides bi-directional traffic across the bi-directional data diode while guaranteeing uni-directional traffic across each data diode.

A data diode is a network link that has been modified to send data only in one direction, thereby creating a uni-directional link.

The impact bi-directional data diodes might have on an IDS detection performance has been analysed.

In addition to looking into how network segmentation and compartmentalisation can be done with bi-directional data diodes we have compared it with traditional security mechanisms such as firewalls and Access-Control Lists (ACLs). It is shown that we can achieve similar results with regard to segmenting the network with both bi-directional data-diodes and traditional security mechanisms.

# Contents

# List of Figures

# List of Tables

# Listings

# 1 Introduction

## 1.1 Topic covered by the project

Most networks are vulnerable to many kinds of attacks on different devices. After an attacker gains a foothold into one of the devices there are two primary ways forwards. The first one is to gain more control of the compromised device through attacks such as privilege escalation, sandbox and VM escape. The second way forward is to gain access to other devices through the network. This thesis looks at how the attackers reach across the network can be limited by compartmentalisation of the network with bi-directional data diodes.

Network based data diodes are network links that have been modified to only send traffic in one direction. One way to achieve this is by using fiber-optic connections where the cable is only connected to the sender on one side and the receiver on the other side [2, 3, 4, 5].

We also look at how the detection rate and performance of an IDS is impacted by the use of data diodes in the network.

## 1.2 Keywords

bi-directional data diode, data diode, networking, network simulation, network design, intrusion detection system, IDS

## 1.3 Brief explanation of data diodes

A data diode is a physical device that only allows the transmission of data in one direction from one device to another. This can be done purely in hardware or by a combination of hardware and software. The name comes from the world of electronics where a diode is an electronic component that only allows the current to travel in one direction.

We know of two types of data diodes that are in use today. The first one is a write blocker which is used to make a forensic image or copy of a hard drive or a USB stick. The second one is a network data diode. Network based data diodes are the focus of this thesis and we explain them more thoroughly in Chapter 4.

### 1.3.1 Bi-directional data diodes

In this thesis we suggest supporting bi-directional traffic on links that are protected by data diodes. This is achieved by adding an additional data diode with the opposite orientation, thus providing two uni-directional links between two nodes. We call the aggregated link with two data diodes a bi-directional data diode.

## 1.4 Problem description

How can we improve network containment and isolation of attacks in data centers and corporate networks? There exist multiple different tools and solutions to provide containment and isolation on the hosts in the data centers such as sandboxes, containers and virtual machines. To contain attacks to a specific host or network segment, we can use network tools such as ACLs, firewalls, and Intrusion Prevention Systems (IPSes). A major limitation with all of these solutions is that they are realised in software in computers or embedded devices and will therefore have some bugs. Some of these bugs may create vulnerabilities that defeat the protection offered by the network tool. An attacker can exploit these bugs to access the protected parts of the network. In addition, misconfiguration of the network tool can add vulnerabilities to the networks.

There exist industrial control systems and high security network segments that use data diodes to ensure uni-directional traffic. Data centers or corporate networks require bi-directional network traffic. Is it possible to replace a traditional bi-directional link in a network with a bi-directional data diode? The goal is to design networks that ensure containment of attacks.

An Intrusion Detection System (IDS) is used to detect attacks and intrusions into different systems such as networks. Are there any performance impact on detection rate or system resource usage for an IDS in a network that uses bi-directional data diodes?

## 1.5 Justification, motivation and benefits

Attacks such as NotPetya and WannaCry used bugs in Windows® to quickly spread through entire companies and encrypt files or destroy hard drives. These attacks show the importance of network containment and isolation [6, 7] both in the corporate world and in data centers.

Data diodes are traditionally used to create a uni-directional link between network segments. This ensures that traffic can only travel in one direction across that link and creates segmentation that is not vulnerable to software bugs or software misconfiguration. One drawback is that special software is needed to support bi-directional protocols such as Transmission Control Protocol/Internet Protocol (TCP/IP).

Most applications use bi-directional communication. A data diode prohibits this and is therefore rarely used in networks that do not have very strict security requirements. Is it possible to add additional data diodes to the network and thus provide bi-directional communication between network segment, and still keep some network segments isolated from each other with the very high security data diodes provide?

## 1.6 Research questions

For this thesis there are two main groups of research questions. The first group looks at how bi-directional data diodes interact with the use of IDSes. The second group of research questions considers how bi-directional data diodes can be used to contain attacks or malware.

The research questions for this thesis are:

1. What is the impact of bi-directional data diodes on IDS performance?

   1. What is the impact on false/true positive/negative rate in networks with bi-directional data diodes?
   2. What is the impact on packet processing time or resources based on IDS location in a network with bi-directional data diodes?
2. How can we limit the propagation of attacks by means of bi-directional data diodes?
   1. Prevent sideways movement of attackers(movement from server to server etc.)?
3. What is the benefit of using bi-directional data diodes over traditional solutions when containing an attack or malware?

## 1.7 Planned contributions

The planned contribution for this thesis is knowledge about how bi-directional data diodes can be used in a network. This includes information about probable common mistakes and issues, such as how stateful firewalls or intrusion detection systems can be placed and configured to work in a network with data diodes.

In addition, we look at how different network designs with bi-directional data diodes provide different strengths and weaknesses such as general performance, ease of configuration, and security. This allows others to implement networks with data diodes to hopefully improve their network security.

During the work with the thesis we will look at how we can simulate data diodes using network simulation tools, and provide instructions on how other people can simulate data diodes using the same tools that we used.

# 2 Choice of methods

To answer our research questions it is necessary to test the IDS performance and the ability of different network designs to contain attacks. We also need to select the scientific methods we use for our experiments. The selection is based on textbooks describing criteria for selecting scientific methods. This chapter therefore consist of two parts; first a discussion on how we can test different network topologies and a conclusion on which method that we have selected, second a discussion about the scientific methodologies.

## 2.1 Network testing

To test different network topologies with data diodes there are three different methods that we can use. The first one is to build the networks using physical hardware. The second one is to simulate everything on a computer and the third one is a combination of the two, where parts of the network are simulated and other parts of it are created with physical hardware. Each method has its own strengths and weaknesses that we discuss in the following sections.

We also need to decide whether to use live traffic or simulated traffic in the experiments.

### 2.1.1 Real hardware

It is not uncommon to use physical hardware when performing experiments on data diodes. These experiments concern the use of a single data diode and the communication between a single receiver and transmitter [2, 3].

For our experiments we would require at a minimum two data diode links, but having more data diodes would increase the number of network topologies that we can test. A physical data diode might be either a modified link or a commercial data diode. The main pros and cons of using real network hardware are:

**Pros:**

- Ensures that we can create data diodes
- Supports real-time testing

**Cons:**

- Supports only a limited set of network designs based on the number of data diode links that we have available
- Cost
  - we might need to buy hardware to create the networks
  - commercially available data diodes are expensive
- Not easily repeatable
- Potential ethical and legal issues such as GDPR if we use live traffic

### 2.1.2 Simulation

Network simulations can be done in multiple different tools to simulate both network designs and network traffic. These tools allow the test to be repeated and give the researcher full control over the network. It is also easy to scale the simulated networks.

Researchers have used purpose built network simulators such as ns-2 [8] used by Chen et. al to determine the best placement for IDSes in a large network [9]. Other researches have used virtual machines and hypervisors such as the study by Aryachandra et. al [10] who looked at the best placement for IDSes in environments with many virtual machines where they used the Proxmox framework. Proxmox is a custom Linux distribution designed to by run as a hypervisor [11]. This shows us that there are at least two different methods of simulating networks. The main pros and cons of using simulations are:

**Pros:**

- Repeatability of the experiment
- Cheap - there exists open source and free network simulation tools
- Flexibility - we are not limited by available hardware when we design the networks

**Cons:**

- Might need to add support for data diodes

### 2.1.3 Combination

The last method is to combine the first and the second method where parts of the network are realised in hardware while the remaining parts are simulated. This should be cheaper than pure hardware testing and has most of the flexibility of simulated solutions at the cost of higher complexity when configuring the network. The main pros and cons of combining hardware and simulation are:

**Pros:**

- Cheaper than using only real hardware
- Ensures some data diode links in the network

**Cons:**

- Not easily repeatable
- Increased complexity
- Potential ethical and legal issues such as GDPR if using live traffic

### 2.1.4 Conclusion

We have shown that all three methods can be used to test how we can use data diodes in a network, but we need to select one method that we shall use for the thesis. The deciding factor when selecting the method is cost and flexibility. Both *real hardware* and *combination* might require us to buy the necessary equipment and limits the number of different network designs that we can test, based on the available hardware. This leaves *simulation* with network simulation tools as the preferred method to perform the experiments for this thesis.

One additional question that we need to answer is what type of simulation to use, a pure network simulator, virtual machines or a combination of both? Using only virtual machines is not a good solution for us since we need to modify the network connections between the Virtual Machines (VMs) and the hypervisor provided router to create data diodes. The networking code in the hypervisor will need to be modified. This is assumed to be too much work to finish in the timeframe of this thesis.

Using a network simulation tool should give us access to modify any aspects of the network and thereby make it easier to create data diodes in the network. The potential issue with this solution is that it might not support generation of every type of network traffic that we might want to test.

A combination of both virtual machines and network simulation is the best solution as it gives us full control over the simulated networks and it allows us to test the network with live traffic from applications and servers. The main disadvantage with this solution is that it is expected to be harder to set up as it requires both the network simulation tool and VMs to be configured to work together. We have therefore selected to start with only a network simulation tool and if we get time we will use the combination of VMs and network simulation tools.

If we use live traffic, we will take care to use traffic that cannot cause any GDPR issues or other legal or ethical issues.

## 2.2 Scientific methodologies

As shown earlier in this chapter, we base our research on simulating network designs. The question is then which scientific method should we use to answer our research questions. The two main categories of research methodologies are quantitative and qualitative research.

Quantitative research focuses on comparing metrics that can be quantified down to numbers such as network performance [12].

Qualitative research, on the other hand, is research that focuses on real life scenarios without simplifying them down to numbers. One type of qualitative research is case studies [12].

Robert Yin mentions in [13] that *how* and *why* questions often lead to the use of a case study, while *what* questions often leads to either exploratory studies or surveys studies. According to Yin in [13] case studies are preferred when we do not control all the variables and the exact same experiment cannot be repeated.

Based on this information we can choose the appropriate research method for our research questions. Our first research question is a *what performance impact* question. A quntitative methodology is appropriate and a number of IDS performance related experiments will be performed to measure the processing time, resource usage and detection performance of an IDS.

Our second research question is formulated as a *how* question. There is nothing in the wording that indicates comparisons of numbers. A qualitative methodology is applicable, and we have chosen to perform a case study where we look at how different network designs using bi-directional data diodes can be used to enforce segmentation and limit the propagation of attacks and/or malware in a network.

The third research question is a *what benefit* type of question. A qualitative methodology is appropriate and we have chosen to use an exploratory study. We discuss the security related benefits and drawbacks of the solutions we compare.

# 3 Related work

Academic studies of a data diode generally falls into one of two categories. The first category is papers on how a data diode works and can be created and how data diodes impact transfer performance. The second category is papers that look at how a data-diode can be used to protect and secure a network segment or a single device. The use cases that they often presents are industrial control systems.

The closest that we have got to find scientific papers that address bi-directional data diodes are papers that presents a data diode with a limited return channel. The return channel is often used to ensure the integrity of the transferred data or to increase the transfer speed.

This chapter presents the literature that we have found to be related to the thesis. The first section present literature that explains key concept of data diodes. The two following sections will present the literature that describes how data diodes can be used in industrial control systems and data diodes with a limited return channel.

## 3.1 Data diodes

Kehe, Fei and Wenchao present in [14] how a data diode can be implemented in a real-time fashion with a single bit being sent back to the transmitter to allow the transmitter to resend the packets if any errors are detected during the transmission. The fact that they claim that the data diode does not add any performance penalty is important when considering the use in data centers with lots of traffic. Two last things worth mentioning from this paper are the use of custom protocols for the transmission over the data diode and that the use of a data diode is a way to transmit data between different security layers without the risk of anything being transmitted the other way [14].

Kim and Na present in [3] how data diodes communication can be improved in terms of both speed and reliability by using modified device drivers. This is however not that part that is interesting for our use. We are interested in how they designed the data diode and any pitfalls that they mention might happen during the process. Their data diode uses a fiber-optical connection where only one of the connectors is connected at each end. The transmitting end is connected to the Tx port while the receiving end is connected to the Rx port, which appears to be the common method of creating data diodes [2, 3, 4, 5]. The primary pitfall that they mention is that people by mistake might connect both of the connectors, thereby removing the physical separation provided by data diodes. Their solution is to modify the driver to discard any received traffic at the transmitting end [3]. In our case firewalls or other network tools can be used to protect against physical misconfiguration.

It is also possible to virtualize a data diode as de Freitas et. al. present in [15]. Their main focus, as with most of the other papers in this category, is how well the data diode performs regarding packet loss and bandwidth. In addition, they look at the deployment time of the virtual data diode. The most relevant point from this paper is that at is that they emulate a diode virtually and provide proof of concept that it works. It might have been an interesting method to use for our experiments. The paper was however released after we already had started to implement our chosen method presented in Chapters 2 and 6.

### 3.2 Data diode use cases

Okhravi, Sheldon, and Haines [4] presents how a data diode can be used to ensure one way communication from a protected process control network to the less protected enterprise network. This is one of the three relevant pieces of information from this paper. The second one is that one needs to consider where one places the data diode as they are incompatible with many existing protocols that require bi-directional communication such as TCP/IP, and that a data diode in itself does not provide any protection. In their own words [4]:

> It is sometimes claimed that data diodes protect the high network against cyber attacks. This, in fact, is not correct. Many cyber exploits do not require a session or bidirectional communication. Often fast propagating worms or malware need just one packet of data to infect a machine. Self expanding malware or quine programs [16] even limits the number of bytes required in the packet [17].

This shows us that it is important to use other tools as well, such as firewalls, and figure out how they might be used together with data diodes to provide better protection.

Barker and Cheese present in [18] different network designs to safely connect a nuclear power plants safety systems to the corporate IT network. They describe how different placement of a single or multiple data diode(s) impacts the reach of an attacker. The big difference between this paper and what we want to do is that they focused on industrial control systems, while we focus on how this might be achieved in normal networks where bi-directional communication is necessary. This might remove some of the problems they faced such as supported protocols, but it could also introduce completely new issues.

U.S. Department of Homeland Security recommend in [19] to use data diodes both to minimise the available attack surface and to compartmentalise the network to stop malware from spreading through the entire industrial control system network.

Schlicher present in [20] how a software data diode can be used to prevent data exfiltration. The interesting part for us is not the software data diode, but that the use case is how a data diode is used in a corporate network. The data diode allows transfer of data into a high security network segment, but does not allow any information to leave the segment. This is a similar use case to the one used in this thesis.

### 3.3 Bidirectional communication with data diodes

In [5] Yum et. al present how one might add a limited reverse connection to improve the user experience of a data diode by allowing a limited set of information to be transmitted back to the sender from the receiver. This is done in a similar manner as in [14]. This allows for, amongst other things, simple diagnostics of the data diode.

During our work on this thesis we have found two companies that create and sell bi-directional data diodes. One of the products was released last year. We believe the other product was released this spring while we worked on this thesis. As far as we can tell, these products are two of their single data diode products assembled in one enclosure [21, 22]. The main use case proposed by the vendors is to use the bi-directional diodes in a critical infrastructure. One data diode provides the traditional monitoring of the infrastructure. The second data diode provides a return channel where an operator can control the infrastructure. Native support of bi-directional protocols is not supported [21].

# 4 Theory

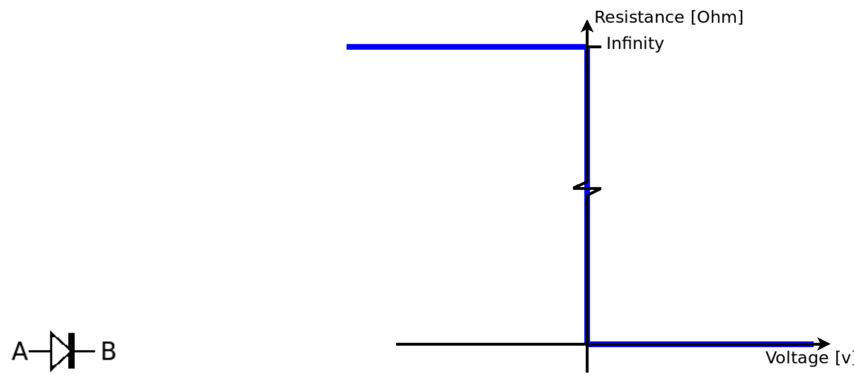
In this chapter we look at how data diodes can be created, and the theoretical strengths and weaknesses regarding the security of the different designs. This will be an indepth look at the different methods to create data diodes. In addition to explain how data diodes work and can be created we will also provide a brief explanation on IDSes and their terminology that we use in this thesis.

## 4.1 Data diodes

Data diodes can mainly be split into two different groups. Group one is data diodes without any sort of return channel. The second group is data diodes with return channels that allows for a limited set of information to be sent back over the diode link, such as if a packet is successfully received or not.

In our use-case there are no security reasons to select either group of data diodes above the other as we want and need full bi-directional communication. There may be performance reasons to chose one over the other but that is outside the scope of this thesis.

So what is a data diode? Let's start by explaining what an ideal electrical diode is as that is where data diodes have got their name from.




(a) The symbol for an electrical diode

(b) Resistance of an ideal electrical diode

Figure 1: An ideal diode

An ideal electrical diode with ports A and B, shown in Figure 1a, is a two-port component that allows current to flow from port A to B, but prohibits any current flowing from B to A. This can also be stated: an ideal diode has zero resistance in the direction from A to B and infinite resistance in direction from B to A, Figure 1b. For a formal definition of an electrical diode, see [23].


(a) The symbol for a data diode

(b) Use of a data diode in a network

Figure 2: Data diode

This can easily be translated to a data network where a data diode, shown in Figure 2a, has the same graphical representation as an electrical diode. It is used to isolate two network segments. In Figure 2b the data diode will allow network traffic to flow from Net A to Net B, but no network traffic can flow from Net B to Net A.

One important thing to note is that it is also possible to implement these traffic regulations by network tools such as firewall and routing rules etc in Net A and Net B. We will discuss and compare the benefits and drawbacks of bi-directional data diodes and network tools in Section 8.1.8.

## 4.2 Data diode properties

The key property of a data diode is that it is uni-directional. This is both its biggest strength and biggest weakness as any bi-directional protocol will not work over a data diode link. This includes TCP/IP, one of the most used Internet protocols, as it is bi-directional. This means that no TCP/IP connections can be established across a link where a data diode is placed.

It is possible to work around this limiting factor by using a proxy on each side of the diode link. The transmitting proxy terminates the Transmission Control Protocol (TCP) session with the transmitter before converting payload and the necessary protocol information to a format suitable to send over the data diode such as User Datagram Protocol (UDP). The receiving proxy converts it back to TCP and terminates the TCP session with the receiver. This also means that the TCP/IP (or other bi-directional protocols) end-to-end data guarantee and flow-control is broken. Data packets that are lost across the data diode link will not be detected by the TCP/IP protocol and will not be re-transmitted since the proxy in the transmitter will acknowledge the packet before it reaches the ultimate destination.

These proxy solutions give the user great control over the data transmitted through the data diode and can be used as an additional layer of protection. A potential weakness with the proxy solution is that the proxy might provide UDP pass through without filtering which then can be used by attackers as a potential entry point into the network [24].

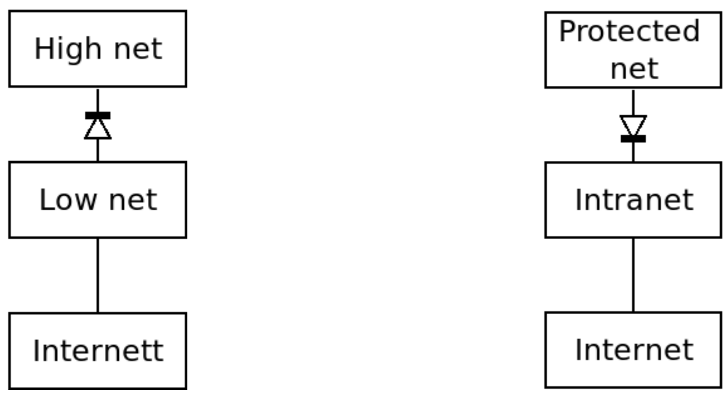
Regardless of whether a data diode is used with a proxy or not, it ensures that no information from the network behind the data diode is transmittable out from that network. This includes information such as IP addresses, OS and program versions and any other data stored in that network.

An important thing to note is that we have not looked at data diode networks that use the proxy solution in this thesis.

### 4.3 Common use cases

There are two common traditional use cases for data diodes. While our focus is on alternative use cases. We present them as they show the strengths and weaknesses of data diodes. The two use cases are closely related and the primary difference between them is the direction of the diode as shown in Figure 3. The two use cases are:

- Prohibit leaking of classified information as seen in Figure 3a.
- Prohibit infiltration of a protected network as seen in Figure 3b.




(a) A data diode prohibiting leaking of classified information

(b) A data diode prohibiting infiltration of a protected network

Figure 3: Common data diode use cases

#### 4.3.1 Prohibit leaking of classified information

This use case is applicable when one wants to keep all the information and data inside a network. A common example is a classified network. The main users are military nets and governmental nets. The data diode is placed between a network containing data with high classification and a network containing data with lower classification. It is oriented such that data can flow from the lower classified net to the higher classified net.

The data diode provides a guarantee that no data can flow from the higher classifed network to the lower classified network across the link proteced by the data diode.

An important thing to note with this use case is that it does not block any malware or attacks that use unidirectional protocols such as UDP to infect machines from entering the classified network [4]. It does however stop the potential malware control server or attacker from receiving any information or files from the classified network.

#### 4.3.2 Prohibit infiltration

This use case is applicable in networks that need a high guarantee that an attack from the Internet cannot reach the protected network. At the same time, the user for example requires the ability to monitor the network and/or equipment connected to the network. Such networks includes critical infrastructure. The data diode is placed before the gateway to the protected network and is oriented such that no data can flow from the Internet to the protected network.

## 4.4 Data diode implementations

In this section we will first present the main principle behind data diode creation, before looking at how that might be implemented over optical and electrical Ethernet.

The main principle behind how a data diode is created is the same regardless of the technology used in the diode link. It is to physically remove the ability to send data in both direction across a link. There are two main methods of doing this. The first one is to modify the Network Interface Controller (NIC) by removing the receiver or transmitter part of the NIC (depending on which end of the connection the NIC is placed). The second, and simpler, method, at least if you make the data diode yourself, is to modify the cable by disconnecting all wires of fibres sending data in one direction. A simple illustration of this is to disconnect either the Tx or Rx fiber over a fiber optic connection. One important thing to note is that as long as we deal with Ethernet it is necessary to manually fill the ARP-table of the NICs that are a part of the link since the response to the Address Resolution Protocol (ARP)-request is either not sent or not received depending on which end of the diode you consider.

### 4.4.1 Optical Ethernet

One of the conceptually simplest methods of creating a data diode is to use an optical Ethernet cable where only one fiber is connected from transmit on one side to receive on the other, thus ensuring that traffic only can flow in one direction.

This is a very simple solution, but not fool-proof, since both fibers can be connected later. This is analogous to a faulty configuration in a router or firewall. Bespoke hardware, where only the transmitter or reciever is present in the assembly, will mitigate this problem and ensure that connecting both fibres will not disable the data diode function of the link.

When using 10 Mbit (10-BaseF, [25] clause 15) or 100 Mbit (100-BaseX, [25] clause 24) optical Ethernet, the transmitter needs to get a carrier detect signal before transmitting data. In a data diode application, the receiver is not connected to the other side, and another source must provide the carrier signal to the transmitter. Malcolm W. Stevens [1] solves this by adding an additional 10Mbit or 100 Mbit Ethernet transmitter that is connected directly to the receiver of the transmitter, as shown in Figure 4

When using Gigabit optical Ethernet (1000-BaseX, [25] clause 36) the additional transmitter does not seem to be needed. Heo and Na describes in [2] how a data diode for Gigabit Optical Ethernet was created by using two NICs and a single fiber.

### 4.4.2 Electrical Ethernet

An electrical Ethernet can also implement the same uni-directional connection by only connecting the transmitter in one end to the receiver in the other. This requires an unique connection between

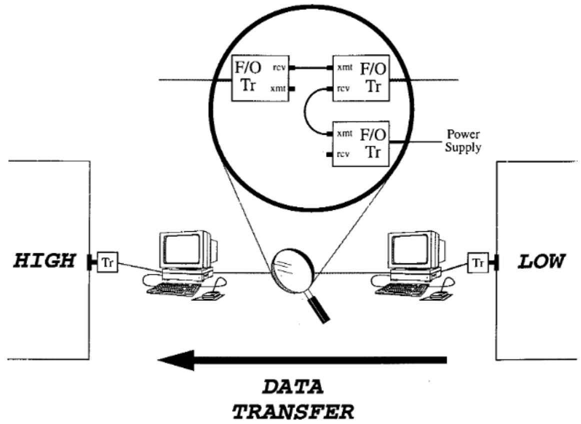



Figure 4: Implementation of a optical Ethernet Data Diode[1]

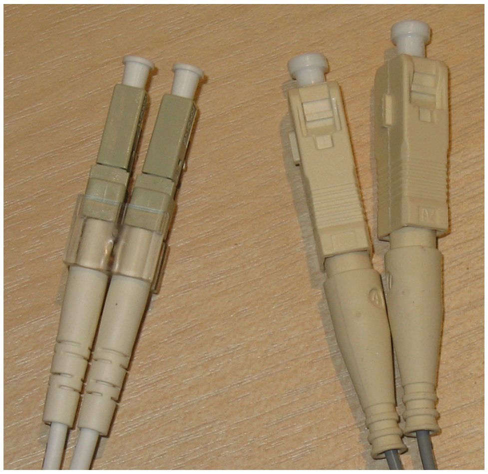
the transmitter in one end of the cable and the receiver on the other. Electrical Ethernet supports a large number of physical interfaces. The most commonly used is Unshielded Twisted Pair (UTP) cat5 cable with four pairs.

10 Mbit (10-BaseT [25], clause 14) and 100Mbit (100-BaseT4 [25], clause 23) electrical Ethernet uses two of the four pairs in the cat5 cable, one for transmit and one for receive. Auto-MDI-X is a process where the NICs detects if a cross-over or straight cable is used. Autonegotiation may also be used to resolve speed and duplex mode.

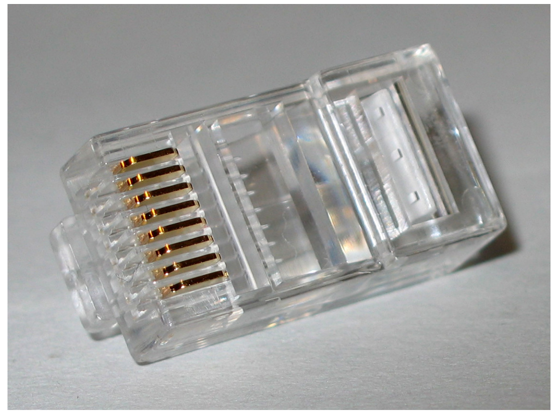
To create a cable that creates a data diode with 10-BaseT or 100-BaseT Ethernet, only one pair must be connected in the connectors. The NICs must be configured to neither use autonegotiation nor auto-MDI-X, and with the same speed and with the correct selection of connector pins for receiver and transmitter.

This solution has the same drawback as a single-fiber solution. The cable may be replaced by a fully configured cable, thus providing a fully functioning duplex link. It might however be harder to see this issue with electrical Ethernet then optical Ethernet as fibers might have two separate connectors while electrical Ethernet has a single connector as shown in Figure 5. Using bespoke NICs with either only transmitter or reciever connected is a mitigation for this problem.

1000-BaseT or faster variant of electrical Ethernet over UTP cable sends and receives simultaneously on all four pairs in the cable, thus making it impossible to implement a data diode by disconnecting some of the pairs in the cable. In addition, IEEE 802.3 [25], requires that the autonegotiation process must be used to decide which end of the link shall be the clock master. Autonegotiation requires a bi-directional connection to work.



(a) Two types of optical Ethernet connectors - LC and SC [26]



(b) Electrical Ethernet connector - RJ45 [27]

Figure 5: Ethernet connectors

## 4.5 IDS

IDS stands for Intrusion Detection System and is a device or application that monitors the system for intrusions and/or attacks. IDSes can be classified by the system they monitor and how they detect intrusions and/or attacks. The different types of systems that an IDS monitors includes hosts ("single computers"), networks, and applications. The two primary detection models are misuse and anomaly detection. Misuse detection uses rules or signatures to detect intrusions and/or attacks, while anomaly detection generates alerts if the traffic deviates too much from the normal traffic pattern. The normal traffic pattern may be either manually or automatically updated.

Due to the nature of the experiments performed in this thesis, where we simulate small networks over short periods of times, we are focusing on network based misuse detection IDSes.

There exists different misuse detection IDSes that each operates in a slightly different manner from each other. We focus on the primary underlying principle instead of the different nuances in how they work. That incoming traffic or activity, in our case packets, is analysed and compared against a set of rules. If the packet matches a rule, then an alert is generated. The rules can instruct the IDS to do other things than generating an alert if a match occurs. These actions include log, pass, drop, and reject. If the latter two are used, the IDS will work as an IPS. Both IDSes and IPSes often includes functionality to look at session streams in addition to single packets. They also often include functionality to detect attacks or attack indicators that it is hard to write rules for, such as port scanning. The IDS needs to look at more than one session or one packet to detect this type of activities.

For this thesis have we chosen to use Snort [28] as our IDS. For more information about this choice see Section 6.1. An example of a Snort rule used in this thesis is shown in Listing 1.

```
alert udp [10.0.3.0/24] any -> any any (msg: "Connection from:
    10.0.3.0/24 detected"; sid:1;)
```

Listing 1: Example Snort rule

This rule generates an alert for each `UDP` packet received from `any` port in the `10.0.3.0/24` IP-address range to `any` port in `any` IP-address, and outputs the following message to the log: `Connection from: 10.0.3.0/24 detected`.

This rule is designed to generate an alert for any UDP packet detected from the `10.0.3.0/24` network. This type of rule is only usable for the cases where no communication from a certain network segment is allowed. This is not a typical Snort rule since it does not use variables for the IP-addresses and uses the keyword `any` in a very liberal manner.

#### 4.5.1 IDS evaluation

When evaluating how good an IDS or a single IDS rule is at correctly classifying and detecting attacks, we use the indicators True Positive Rate (TPR), False Positive Rate (FPR), True Negative Rate (TNR) and False Negative Rate (FNR). True positive, false negative etc. are defined in Section 10.2. TPR and FPR are calculated using

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} = 1 - \mathrm{FNR}$$

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}} = 1 - \mathrm{TNR}$$

where TP = True Positives, FN = False Negatives, FP = false positives, and TN = True Negatives.

The TPR is the fraction of the positives that is detected as positive. Similarily, the FPR is the fraction of the negatives that is detected as positive.

The best IDS and IDS rule has as large a TPR as possible and as small a FPR as possible.

The resource utilisation is also a relevant metric for us when evaluating hypothesis 2.

# 5 Hypothesis

The research questions from Section 1.6 can be split into two groups, the first one focuses on IDS detection performance in a network that uses data diodes while the second group focuses on network design of networks with data diodes.

We have two hypotheses regarding the detection performance for an IDS depending on the relative placement of IDS and bi-directional data diode.

**Hypothesis 1:** There is little or no difference on the detection performance of an IDS placed in a part of the network behind a bi-directional data diode as it needs to process the same amount of packets as in a traditional network.

**Hypothesis 2:** An IDS monitoring the traffic over one of the links comprising a bi-directional data diode should have improved detection performance or reduced resource usage compared to an IDS monitoring a traditional link. The IDS only needs to monitor approximately half the traffic since it only monitors one direction of the connection instead of both directions.

We have opted to not create any hypotheses for our second group of research questions which focuses on network design. The reasoning behind this is that we are performing a case study to answer these research questions. We look at and discuss the benefits and drawbacks of different network designs where bi-directional data diodes are a main feature. The results of these test cannot be simplified down to simple yes/no answer or a comparison of numbers that tells us if our hypotheses are correct.

# 6 Implementation

This chapter describes the rationale for selecting the tools used, how we implemented data diodes in ns-3, the problems that we encountered and how we solved them. We also describe the tests that were performed.

## 6.1 Selection of tools

We have selected both a network simulator and IDS to use for our experiments. The process and rationale for the selection is presented below.

### 6.1.1 Network simulation

We have looked for a network simulation tool that has support for data diodes or the support for data diodes should easily be added. In Chapter 3, [8] used ns-2, while in our report for IMT4205, [29], we found a project that added support for fiber optic connections to ns-3. We also discovered that ns-3 has support for disabling the transmitter or receiver individually in a CSMA interface. This allowed us to emulate a data diode. ns-3 was therefore chosen as our network simulation tool for this thesis.

### 6.1.2 Description of ns-3

ns-3 is a discrete event network simulator that is created to allow researchers to simulate networks for network research [30]. ns-3 is a complete rewrite of ns-2 and focuses on improving certain aspects of ns-2 such as the core architecture, software integration, models, and educational components. It should be noted that ns-3 is not backward compatible with ns-2 [31]. The goal of ns-3 is:

> The goal of the ns-3 project is to develop a preferred, open simulation environment for networking research: it should be aligned with the simulation needs of modern networking research and should encourage community contribution, peer review, and validation of the software [30].

### 6.1.3 Intrusion Detection System (IDS)

The selection of an IDS was a simple process. We wanted to use an IDS that we were familiar with from earlier projects and courses. This left us with two choices: Snort [28] and Suricata [32]. We chose Snort since we have slightly more experience with using it.

## 6.2 Test environment

To test our first research question for this thesis we have created two very similar networks in ns-3. The only difference is that one network, shown in Figure 6, uses data diodes to connect the different parts of the network while the other one, shown in Figure 7, uses traditional connections.

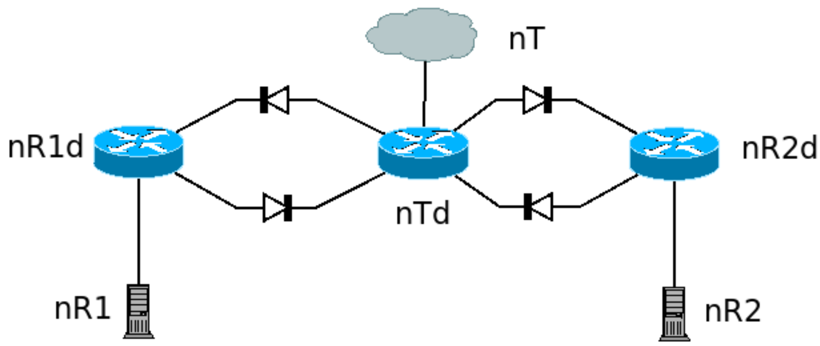



Figure 6: Simulated test network

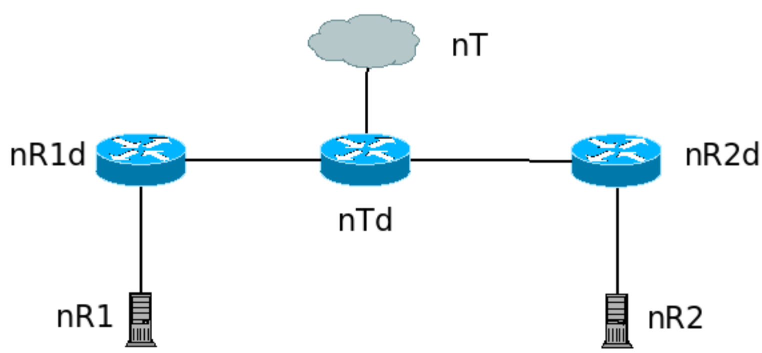



Figure 7: Simulated test network in traditional mode

The nets consists of following nodes:

**nT** representing the Internet and everything outside the test network..

**nTd** representing the edge router and gateway to our test network.

**nR1d** representing routers for the nR1 networks.

**nR2d** representing routers for the nR2 networks.

**nR1** representing computers in the nR1 networks.

**nR2** representing computers in the nR2 networks.

The test uses the ns-3 `UdoEcho` application to generate test traffic in the network. The `UdpEchoClient` sends UDP packets and logs both when a packet is sent and when a reply is received. The `UdpEchoServer` receives UDP packets and echoes them back to the sender. It also logs when packets are sent and received.

nT has two `UdpEchoClient` instances to send traffic both to nR1 and nR2. nR1 has one `UdpEchoClient` instance to send traffic to nR2 and vica versa for nR2. Both nR1 and nR2 have an `UdpEchoServer` instance as well. This configuration allows us to test if the various leaf nodes can send and receive packets to each other and provides traffic that can be captured to test the IDS performance.

The baseline routing environment is:

- nR1: Static route for local network to nR1d and default route to nR1d.
- nR2: Static route for local network to nR2d and default route to nR2d.
- nT: Static route to nR1 and nR2 networks to nTd.
- nR1d: Static routing via transmit diode to nTd for packets to the nT network, and default route via the same interface for packets to other networks.
- nR2d: Static routing via transmit diode to nTd for packets to the nT network, and default route via the same interface for packets to other networks.
- nTd: Static routing via transmit diode to nR1d and nR2d.

The routing environment is changed in some of the experiments.

In Chapter 2 we mentioned the possibility of combining network simulations with VMs. ns-3 supports this through the `TapBridge` interface [33, 34]. We tried to get this to work but ran out of time before we got it working. We therefore only use traffic generated from the simulation.

## 6.3 Test description

The research questions are answered by two different tests.

The first test forms the basis to answer our second and third research question. The results are used in our case when discussing the case studies and comparison of traditional protection methods against network designs with bi-directional data diodes. The test is performed by having each `UdpEchoClient` send one UDP packet to nR1 and nR2. The terminal output from the simulator

| item | description |
|---|---|
| ns-3 | Version 3.29 |
| CPU | Intel Xeon E5-1650 V3 |
| RAM | 16 GB |

Table 1: Simulation environment used

is analysed to show which leaf nodes can receive and/or transmit packages to each other. The clients are started with one second intervals to make it easier to parse the results of the test. The routing rules are changed between tests to see how that impacts the results.

The second test provides data to answer our first research question. We need large amounts of traffic to evaluate the IDS performance. The simulation can be configured from the commandline to send more packets per `UdpEchoClient` and to instantiate multiple instances of the `UdpEchoClient` per leaf node.

The simulation generates packet captures in pcap format for all links to nR1d and nR2d. The packet captures are then evaluated by Snort with some simple rules. The detection results are stored and analysed to determine the true or false positive and negative rate for the different simulations. We also recorded and analysed the `Run time for packet processing` and some of the packet processing rates of Snort to use as indicators of Snort resource usage on the system.

## 6.4 ns-3 environment

For our experiments we used version 3.29 of ns-3. This is the most recent version at the start of our experiment. It is run directly on the host.

The simulation environment and hardware used for our test is listed in Table 1

### 6.4.1 Creating a data diode in ns-3

Creating a data diode in ns-3 proved to be both easier and harder than first expected. The easy part is that the Carrier-sense multiple access (CSMA) network device in ns-3 has configuration options that allow the user to disable either the send or receive part of any CSMA interface in effect creating a data diode

The harder part is that this conversion from a normal link to a diode link breaks the underlying ARP protocol used to map Internet Protocol (IP) addresses to Media Access Control (MAC) addresses. Our first tests, with only one data diode from nTd to nRd in Figure 8, were unsucsessfull as the ARP reply from nRd to nTd was blocked by our data diode. This proves that our simulated data diode works as we were able to transmit packets from nTd to nRd but not the other way around. The solution for this is to pre-populate the ARP-table for the transmitter diode. ns-3 supports this and once we managed implement this in our code were we able to successfully transmit packets across an uni-directional data diode.

These issues are not unique to simulations in ns-3. We expect that both the routing table and ARP-table needs to be manually configured when using custom data diodes. This is expected as

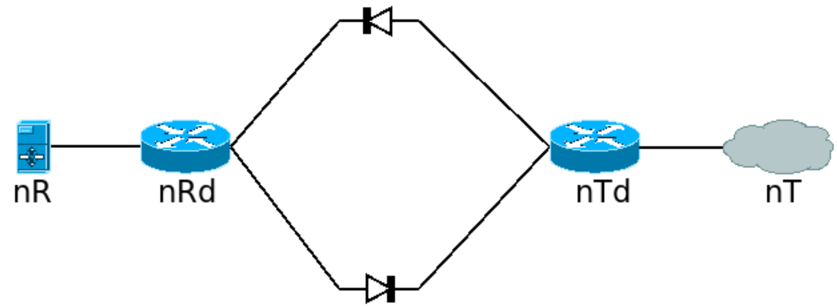



Figure 8: A simple network with two data diodes

ARP is a bi-directional protocol and many routing protocols need bi-directional communication to work and would therefore not work over a data diode as shown in Chapter 4. Buying commercially available data diodes is expected to take care of these issues for the user of the system.

## 6.5 Creating a bi-directional data diode in ns-3

Most of the issues encountered when creating bi-directional data diodes in ns-3 were caused by limitations in the network simulation tools. Event if they are created by limitations inside ns-3, it is not improbable that one might hit similar issues when creating a bi-directional data diode.

We created a function to automate the creation of bi-directional data diodes in ns-3. All of these issues are not directly related to that function, although we chose to solve some of them in the function. These issues ranged from limitations with some of the functions provided by ns-3, to misunderstandings regarding the order method calls need to be performed in ns-3.

The first issue that we encountered when we tried to create bi-directional data diodes as illustrated in Figure 8 was that we could not use the ns-3 method, `Ipv4GlobalRoutingHelper::PopulateRoutingTables()` to auto-populate the routing tables. This method in ns-3 does not work when there are loops in the network. A bi-directional data diode does not really create a loop, but the ns-3 code does not understand this subtlety. The work around for this is to change from automatic routing to static routing. Note that when we created uni-directional data diode links, described in Section 6.4.1, `Ipv4GlobalRoutingHelper::PopulateRoutingTables()` worked.

Another issue that we ran into is that `CsmaNetDevice`, the network device and interface from ns-3 that we use for our network connection, does not support full duplex mode. The communication happens in half duplex mode [35]. This causes packet drops as shown in Table 2. It should be noted that the packet drop is larger in the columns labelled 'normal', i.e. when the bi-directional data diode is replaced by a normal bi-directional link. This is expected since the packet drop is caused by collisions when both sides simultaneously transmit a packet across the link, and since a single data diode link can only transmit data in one direction avoids any collisions. An interesting thing to note is that the number of lost packets is the same regardless of how many packets that we originally

| | nR1 - nR2 data diode | nR1 - nR2 normal | nR2 - nR1 data diode | nR2 - nR1 normal |
|---|---|---|---|---|
| transmitted packets: | 100 | 100 | 100 | 100 |
| received packets: | 95 | 82 | 96 | 85 |
| lost packets: | 5 | 18 | 4 | 15 |
| transmitted packets: | 10000 | 10000 | 10000 | 10000 |
| received packets: | 9995 | 9982 | 9996 | 9985 |
| lost packets: | 5 | 18 | 4 | 15 |
| transmitted packets: | 100000 | 100000 | 100000 | 100000 |
| received packets: | 99995 | 99982 | 99996 | 99985 |
| lost packets: | 5 | 18 | 4 | 15 |
| transmitted packets: | 100000 | 100000 | 100000 | 100000 |
| received packets: | 100000 | 100000 | 100000 | 100000 |
| lost packets: | 5 | 18 | 4 | 15 |

Table 2: Packet loss from ns-3

sent. We tentatively conclude that the packet loss happens at the beginning of the simulation. The packet loss does not impact the performance result from our IDS testing as we know how many packets that actually arrived at the intended location and should generate an alert in our IDS

## 6.6 Explanation of CreateDiode function

In this section we present how our `CreateDiode` function works by looking at three excerpts of the code and explain the purpose of the code segments. The full source code for the `test-network` program including the complete `CreateDiode` function can be found in Appendix B.

```
// A function to create a diode connection
void
CreateDiode (Ptr<Node> sender,
    Ptr<Node> receiver,
    char const* adress,
    char const* subnetMask,
    char const* baseAdr,
    Ipv4StaticRoutingHelper* ipv4RoutingHelper,
    Ipv4Address destAdr,
    Ipv4Mask destMask,
    bool pcap=false
    )
{
// Create the network link
CsmaHelper csma;

NodeContainer nodes = NodeContainer (sender, receiver);
```

```
18
19 // Create the "network interfaces" and add them to the appropriate nodes
20 NetDeviceContainer diodes;
21 diodes = csma.Install(nodes);
22
23 // Configure the interfaces as diodes
24 Ptr<CsmaNetDevice> diodeS = DynamicCast<CsmaNetDevice> (diodes.Get(0));
25 diodeS->SetReceiveEnable (false);
26 diodeS->SetSendEnable (true);
27
28 Ptr<CsmaNetDevice> diodeR = DynamicCast<CsmaNetDevice> (diodes.Get(1));
29 diodeR->SetReceiveEnable (true);
30 diodeR->SetSendEnable (false);
```

Listing 2: CreateDiode function part 1

The first part of the function creates the network interface on the sender and receiver node and creates the media or channel that the communication happens over. This is shown on line 13 to 21 in Listing 2. Line 24 to 30 show the conversion from a normal network interface to a data diode interface. The receiver is disabled on the sending node and the transmitter is disabled on the receiving node.

```
// Variables used for static routing
Ipv4Address destIntAdress = interfacesDiodes.GetAddress(1);
uint32_t numInterface = diodeS->GetIfIndex();

Ptr<Ipv4> ipv4S = sender->GetObject<Ipv4> ();

// Use static routing for the diodes hopefully allowing for "loops"
Ptr<Ipv4StaticRouting> staticRouteT = ipv4RoutingHelper->GetStaticRouting (ipv4S);
staticRouteT->AddNetworkRouteTo (destAdr, destMask, destIntAdress, numInterface);
// Comment out to disable deafult routes
staticRouteT->SetDefaultRoute (destIntAdress, numInterface);
```

Listing 3: CreateDiode function part 2

The second part of the code, Listing 3, adds a static route to the `receiver` node and a default route entry, also to the `receiver` node, to the routing table in the `sender` node. This is only necessary if we have more than one direct route between two nodes in ns-3 as discussed in Section 6.5. This might also be necessary if there are more complex loops between nodes in ns-3, but we did not test that.

The reason we add both a static route entry and default route is that we want to be able to test what happens when we disable the default route. When the code includes both entries, it is simple to disable one of them by commenting out the relevant line in the code before running a specific test.

```
  // Manually fill the ARP cache of the transmit node Ptr<ArpCache>
```

```
2    arpT = CreateObject<ArpCache> (); arpT->SetAliveTimeout (Seconds
3    (3600 * 24)); // Keep the ARP table entry for one day...
4
5    ArpCache::Entry * entry = arpT->Add (interfacesDiodes.GetAddress(1));
6    entry->SetMacAddress(Mac48Address::ConvertFrom(diodeR->GetAddress()));
7    entry->MarkPermanent ();
8
9    // Add the cache to the transmit node
10   std::pair<Ptr<Ipv4>, uint32_t> returnValue = interfacesDiodes.Get(0);
11   Ptr<Ipv4> ipv4 = returnValue.first;
12   uint32_t index = returnValue.second;
13   Ptr<Ipv4Interface> diodeT = ipv4->GetObject<Ipv4L3Protocol> ()->GetInterface (ind
14   arpT->SetDevice(diodeS, diodeT);
15   diodeT->SetAttribute("ArpCache", PointerValue(arpT));
```

Listing 4: CreateDiode function part 3

The third part shows how the arp-cache is manually filled. This is necessary since the data diode link prevents the ARP response to reach its destination. Line 1 to 7 in Listing 4 adds the MAC adress of the receiving node to the ARP cache while line 9 to 16 adds the cache to the transmitter node.

## 6.7 Running Snort

The command we use when we run Snort is:

```
snort -A console -k none -c <snort-config file> -r <pcap file>,
```

where `-A` tells Snort which alert-mode it should use, `-k none` tells Snort to ignore the checksum in the pcap, `-c` and `-r` tells Snort which configuration file and pcap file that it should use.

The `-k none` is needed since ns-3 does not include valid checksums in the pcap files according to Snort and Wireshark.

# 7 Results

We have performed two primary tests, the first one looks at how changes to routing tables and network design impacts the ability of the network to route traffic between different parts of the network. The second test focuses on IDS performance with regards to runtime and TPR and FPR.

## 7.1 Networking and routing

This section presents our findings on how changes to the routing table on the network nodes with the diode links changes the behavior of the network. The focus is on preventing communication between sections of the network that have no need for intercommunication with each other. Network design is also an important part of this discussion that we will continue with in Chapter 8.

The testing was performed on the network described in Section 6.2 and illustrated in Figure 6. We performed three tests where we changed the routing table of nR1d and nR2d to test the following scenarios:

**Default route** No changes to the routing table shown in Section 6.2

**No default route** Removed the default route from nR1d and nR2d

**Black holing** Added a static entry to the routing table of nR1d and nR2d where traffic to the nR2 and nR1 network is routed to the receiving diode instead of the transmitting diode

How these changes were performed is described later in this section.

The results of these experiments are presented in Table 3, where each marked field indicates successful communication from nX to nY. These results are derivied from the simulation logs as described in Section 7.1.1.

For all the tests nT is able to both transmit and receive packets to/from nR1 and nR2. For the default route test nR1 and nR2 are able to transmit and receive packets to/from each other. The no default route tests makes them unable to transmit or receive packets to/from each other. Blackholing nR2 on the nR1d router allows nR2 to transmit packets to nR1, but nR2 is, however, unable to receive any packets from nR1. In other words nR1 is able to receive packets from nR2, but unable to transmit any packets to nR2. Blackholing nR1 on the nR2d router gets similar results. nR1 is able to transmit packets to nR2 while nR2 is unable to transmit packets to nR1.

The output from ns-3 that we base Table 3 on can be found in Appendix A.

| Configuration | nT – nR1 | nR1 – nT | nT – nR2 | nR2 – nT | nR1 – nR2 | nR2 – nR1 |
|---|---|---|---|---|---|---|
| Default route | x | x | x | x | x | x |
| No default route | x | x | x | x | | |
| Blackhole nR1 on nR2d | x | x | x | x | x | |
| Blackhole nR2 on nR1d | x | x | x | x | | x |

Table 3: Results of connection testing

| Node | Interface | IP-address |
|---|---|---|
| nT | 1 | 10.0.1.2 |
| nTd | 1 | 10.0.1.1 |
| | 2 | 192.168.0.1 |
| | 3 | 192.168.0.4 |
| | 4 | 192.168.0.5 |
| | 5 | 192.168.0.8 |
| nR1 | 1 | 10.0.2.2 |
| nR1d | 1 | 10.0.2.1 |
| | 2 | 192.168.0.2 |
| | 3 | 192.168.0.3 |
| nR2 | 1 | 10.0.3.2 |
| nR2d | 1 | 10.0.3.1 |
| | 2 | 192.168.0.6 |
| | 3 | 192.168.0.7 |

Table 4: Overview of each nodes interfaces and IP-addresses

### 7.1.1 Understanding ns-3 simulation output

```
At time 1s client sent 1024 bytes to 10.0.2.2 port 9
At time 1.00101s server received 1024 bytes from 10.0.1.2 port 49153
At time 1.00101s server sent 1024 bytes to 10.0.1.2 port 49153
At time 1.01202s client received 1024 bytes from 10.0.2.2 port 9
At time 2s client sent 1024 bytes to 10.0.3.2 port 9
At time 2.00101s server received 1024 bytes from 10.0.1.2 port 49154
At time 2.00101s server sent 1024 bytes to 10.0.1.2 port 49154
At time 2.00602s client received 1024 bytes from 10.0.3.2 port 9
At time 3s client sent 1024 bytes to 10.0.3.2 port 9
At time 4s client sent 1024 bytes to 10.0.2.2 port 9
At time 4.00001s server received 1024 bytes from 10.0.3.2 port 49153
At time 4.00001s server sent 1024 bytes to 10.0.3.2 port 49153
```

Listing 5: ns-3 example output

Listing 5 shows the output from our blackhole nR2 on R1d output run of ns-3, and we will use it to explain how we got to the results shown inTable 3.

For the routing tests we have delayed the start of each `UdpEchoServer` with one second and only one UDP packet is sent to each destination. This configuration produces small logs that are easy to understand. Both `UdpEchoServer` and `UdpEchoClient` outputs a timestamped message with information about destination IP address when it sends a packet and source IP-address when it receives a packet. An overview of which IP-address that corresponds to which node is found in Table 4. The output can be split into the following sections:

- 1 to 2 seconds: Communication between nT and nR1. The log shows bi-directional communication.
- 2 to 3 seconds: Communication between nT and nR2. The log shows bi-directional communication.
- 3 to 4 seconds: Communication between nR1 and nR2. The log shows no communication from nR1 to nR2.
- 4 to 5 seconds: Communication between nR2 and nR1. The log shows communication from nR2 to nR1 but no communication from nR1 to nR2.

A similar analysis of the remaining logs in Appendix A results in Table 3.

### 7.1.2 No default route

To disable the default route of the diode link it is enough to comment out line 118 in Listing 15 from the `CreateDiode` function:

```
117   // Comment out to disable default routes
118   staticRouteT->SetDefaultRoute (destIntAdress, numInterface);
```

Listing 6: No default route

This changes the behavior of the router to only transmit data that has a destination address that matches the static route table entry added when creating the diode link.

### 7.1.3 Blackholing

To blackhole nR2 on nR1d it is necessary to comment in line 248 in Listing 7.

```
246 Ptr<Ipv4StaticRouting> staticRoutenR1d = ipv4RoutingHelper.GetStaticRouting (ipv4nR1d);
247 // Comment in to enable blackholing of nR2 on nR1d
248 // staticRoutenR1d->AddNetworkRouteTo (Ipv4Address ("10.0.3.0"), Ipv4Mask ("/24"), 2);
249
250 Ptr<Ipv4StaticRouting> staticRoutenR2d = ipv4RoutingHelper.GetStaticRouting (ipv4nR2d);
251 // Comment in to enable blackholing of nR1 on nR2d
252 // staticRoutenR2d->AddNetworkRouteTo (Ipv4Address ("10.0.2.0"), Ipv4Mask ("/24"), 2);
```

Listing 7: Black holing

The modification routes the traffic that goes from nR1d to nR2 to the receiving diode instead of the transmitting diode thereby sending it to nowhere.

The modification is similar when nR1 is blackholded on nR2d. Line 252 in Listing 7 must be commented in.

## 7.2 IDS performance

To answer our hypotheses we ran two different sets of tests. The first one focuses on detection performance while the second one focuses on the resource usage of Snort.

We start by describing how to read the results before we explain potential outliers or other strange things in our results.

### 7.2.1 Understanding the Snort results

Tables 5 to 7 are used to show how well Snort detects our attacks. Each column represents both one capture location and one `UdpEchoClient`. One example is that the columns labelled with *nR1* or *nR1d* show the numbers reported from the `UdpEchoClient` on nR1 and the columns labelled with *nR1d diode* reports the results of Snort running on the packet capture from the receive diode link on nR1d from the nTd. The columns labelled with *diode* are from simulations using the network configuration with bi-directional data diodes as shown in Figure 6. The columns labelled with *normal* are from simulations that are not using any data diodes as shown in Figure 7.

The rows are divided into groups of four for each run that we performed. The first line, labelled tx nPackets, is the number of packets the corresponding `UdpEchoClient` generates. The second line, labelled rx nPackets, is the number of packets the corresponding `UdpEchoClient` recieves. The third line is the number of alerts reported by Snort. The fourth line is the total number of packets analysed by Snort in that run.

Table 5: Initial results from testing Snort

| | nR1 diode | nR1 normal | nR2 diode | nR2 normal |
|---|---|---|---|---|
| tx nPackets | 100 | 100 | 100 | 100 |
| rx nPackets | 95 | 82 | 96 | 85 |
| Snort Alerts | 95 | 82 | 96 | 85 |
| Total packets: | 578 | 507 | 578 | 529 |
| tx nPackets: | 1 000 | 1 000 | 1 000 | 1 000 |
| rx nPackets | 995 | 982 | 996 | 985 |
| Snort Alerts: | 995 | 982 | 996 | 985 |
| Total packets: | 5 978 | 5 907 | 5 978 | 5 929 |
| tx nPackets: | 10 000 | 10 000 | 10 000 | 10 000 |
| rx nPackets | 9 995 | 9 982 | 9 996 | 9 985 |
| Snort Alerts: | 9 995 | 9 982 | 9 996 | 9 985 |
| Total packets: | 59 978 | 59 907 | 59 978 | 59 929 |
| tx nPackets: | 100 000 | 100 000 | 100 000 | 100 000 |
| rx nPackets | 99 995 | 99 982 | 99 996 | 99 985 |
| Snort Alerts: | 99 995 | 99 982 | 99 996 | 99 985 |
| Total packets: | 599 978 | 599 907 | 599 978 | 599 929 |

### 7.2.2 Snort detection performance

The tests shown in Table 5 show less than half the number of Snort alerts compared to Table 6. This is because we changed the Snort rule to also detect return traffic.

Another difference is seen in the rx nPackets rows. The initial tests had trouble with packet drop, as explained in Section 6.5. When we in test 2 and forward replaced the random interval between packets sent from the `UdpEchoClient` instances with a fixed interval, dropped packets is no longer observed in Tables 6 and 7.

When we increase the number of packets enough, packet drop caused by half duplex CSMA was again observed. One example is shown in Listing 9. In this test 106 000 packets are sent towards the link where the capture occurs. 94 packets are lost and the remaining 105 906 packets were captured and subsequently analysed by Snort.

### 7.2.3 Snort performance

To get a better understanding of the Snort performance we analysed the Snort packet processing time and packet processing rate, as reported by Snort.

When processing data read from a packet dump file (.pcap), Snort does not use the embedded timing information to delay the internal processing. A 13 second long packet dump is processed by Snort in less than two seconds. We have not been able to create a test that produces more packets per seconds that the reported packet rate from Snort. That Snort does not use the embedded timing information to delay packets, indicates that Snort also will not drop packets if packets arrive 'faster'

Table 6: Results from running Snort on nR1 and nR2

| | nR1 diode | nR1 normal | nR2 diode | nR2 normal |
|---|---|---|---|---|
| tx nPackets | 100 | 100 | 100 | 100 |
| rx nPackets | 100 | 100 | 100 | 100 |
| Snort Alerts | 200 | 200 | 200 | 200 |
| Total packets: | 604 | 604 | 604 | 604 |
| tx nPackets | 1 000 | 1 000 | 1 000 | 1 000 |
| rx nPackets | 1 000 | 1 000 | 1 000 | 1 000 |
| Snort Alerts | 2 000 | 2 000 | 2 000 | 2 000 |
| Total packets: | 6 004 | 6 004 | 6 004 | 6 004 |
| tx nPackets | 10 000 | 10 000 | 10 000 | 10 000 |
| rx nPackets | 10 000 | 10 000 | 10 000 | 10 000 |
| Snort Alerts | 20 000 | 20 000 | 20 000 | 20 000 |
| Total packets: | 60 004 | 60 004 | 60 004 | 60 004 |
| tx nPackets | 100 000 | 100 000 | 100 000 | 100 000 |
| rx nPackets | 100 000 | 100 000 | 100 000 | 100 000 |
| Snort Alerts | 200 000 | 200 000 | 200 000 | 200 000 |
| Total packets: | 600 004 | 600 004 | 600 004 | 600 004 |

Table 7: Results from running Snort on nR1d and nR2d using the diode recieve pcap

| | nR1d diode | nR1d normal | nR2d diode | nR2d normal |
|---|---|---|---|---|
| tx nPackets | 100 | 100 | 100 | 100 |
| rx nPackets | 100 | 100 | 100 | 100 |
| Snort Alerts | 200 | 200 | 200 | 200 |
| Total packets: | 300 | 604 | 300 | 604 |
| tx nPackets: | 1 000 | 1 000 | 1 000 | 1 000 |
| rx nPackets | 1 000 | 1 000 | 1 000 | 1 000 |
| Snort Alerts: | 2 000 | 2 000 | 2 000 | 2 000 |
| Total packets: | 3 000 | 6 004 | 3 000 | 6 004 |
| tx nPackets: | 10 000 | 10 000 | 10 000 | 10 000 |
| rx nPackets | 10 000 | 10 000 | 10 000 | 10 000 |
| Snort Alerts: | 20 000 | 20 000 | 20 000 | 20 000 |
| Total packets: | 30 000 | 60 004 | 30 000 | 60 004 |
| tx nPackets: | 100 000 | 100 000 | 100 000 | 100 000 |
| rx nPackets | 100 000 | 100 000 | 100 000 | 100 000 |
| Snort Alerts: | 200 000 | 200 000 | 200 000 | 200 000 |
| Total packets: | 300 000 | 600 004 | 300 000 | 600 004 |

than Snort can process packets.

The first metric that we look at when measuring the Snort performance is *Pkts/sec*. which we might use to calculate a theoretical FPR as we were unable to get Snort to drop packets when analysing pcaps.

```
===================================================================
Run time for packet processing was 1.333 seconds
Snort processed 604 packets.
Snort ran for 0 days 0 hours 0 minutes 1 seconds
   Pkts/sec:          604
===================================================================
```

Listing 8: Snort run time 1

Listing 9 shows that Snort analysed 105 906 packets in 1.322 s. Listing 8 shows shows that Snort analysed 604 packets in 1.333 s. This indicates that the run time reported by Snort is not very reliable when the number of packets is small, i.e. below 100 000 packets. Other reasons for inaccuracies in the reported run-time might be different system load when we ran Snort, and that we did not run it on a dedicated system.

While the number of packets analysed per second is less important than the FPR/TPR and FNR/TNR of the system, it is still an important indicator as it gives us information about how many packets the system can analyse per second before it starts to drop packets, which then might influence the FPR/TPR and FNR/TNR of the system. Snort dropping packets can however be mitigated by running Snort on a more powerful system or by splitting it across multiple machines.

```
===================================================================
Run time for packet processing was 1.322 seconds
Snort processed 105906 packets.
Snort ran for 0 days 0 hours 0 minutes 1 seconds
   Pkts/sec:       105906
===================================================================
```

Listing 9: Snort run time 2

In addition to look at the number of analysed packets and *Pkts/sec*, we also look at the total Snort packet processing time to help us answer our second hypothesis. These results are shown in Table 8 and Figure 9. The format of Table 8 is as follows: the first column shows us the number of packets transmitted from each `UdpEchoClient` while the second to fifth column shows us how long it took Snort to process all the packets captured at the indicated link.

The reported packet processing time from the runs where each `UdpEchoClient` transmits less than 10 000 packets is close to identical. The reason for this seems to be that Snort has a minimum reported packet processing time of around 1.3 second for our test system.

Our results for the runs where each `UdpEchoClient` transmits 50 00 to 100 000 packets, shows us that when Snort monitors a link in a bi-directional data diode uses approximately two seconds less to analyse all the packets when compared to monitoring a traditional link.

Table 8: Snort packet processing time

| sent packets | nR1d diode | nR1d normal | nR2d diode | nR2d normal |
|---|---|---|---|---|
| 100 | 1.1523 | 1.399 | 1.339 | 1.387 |
| 1 000 | 1.313 | 1.387 | 1.375 | 1.417 |
| 10 000 | 1.273 | 1.427 | 1.300 | 1.432 |
| 50 000 | 2.403 | 4.507 | 2.246 | 5.557 |
| 60 000 | 2.384 | 7.671 | 3.463 | 7.587 |
| 70 000 | 3.455 | 2.470 | 3.526 | 6.664 |
| 80 000 | 4.653 | 6.548 | 4.507 | 6.712 |
| 90 000 | 5.612 | 7.812 | 4.489 | 7.685 |
| 100 000 | 5.523 | 8.844 | 5.531 | 11.1169 |

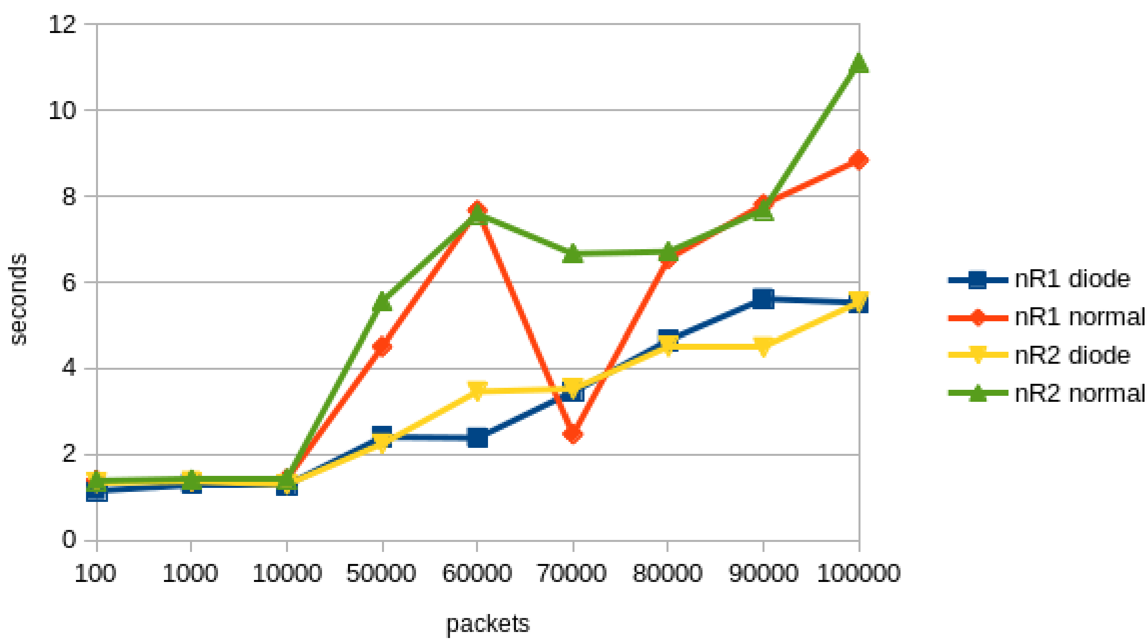



Figure 9: Graph of the results shown in Table 8

The big outlier of the results is the datapoint of the *70 000* packets run of *nR1d normal* where the reported packet processing time is below three seconds. This is less than the reported packet processing time for both the diode links for the same amount of transmitted packets. We were unable to figure out the reason for this. The results are repeatable. Re-running this pcap multiple times in Snort and recreating it in ns-3 provides similar results, all around 2 and a half second.

# 8 Discussion

In this chapter we explain, analyse and discuss the results from Chapter 7. We apply the results to a number of network designs. The IDS performance is also analysed.

## 8.1 Network design and routing

In this section we look at how the use of bi-directional data diode links in different network designs limits the connectivity within the network. The goal is to find network designs that can reduce the possible movement of an attacker inside a network. The same designs will also limit the spreading of malware through the entire network. We also discuss the possibility of recreating the same designs using traditional tools such as firewalls, ACL and routing configuration, and look at the strength and weaknesses of the different solutions.

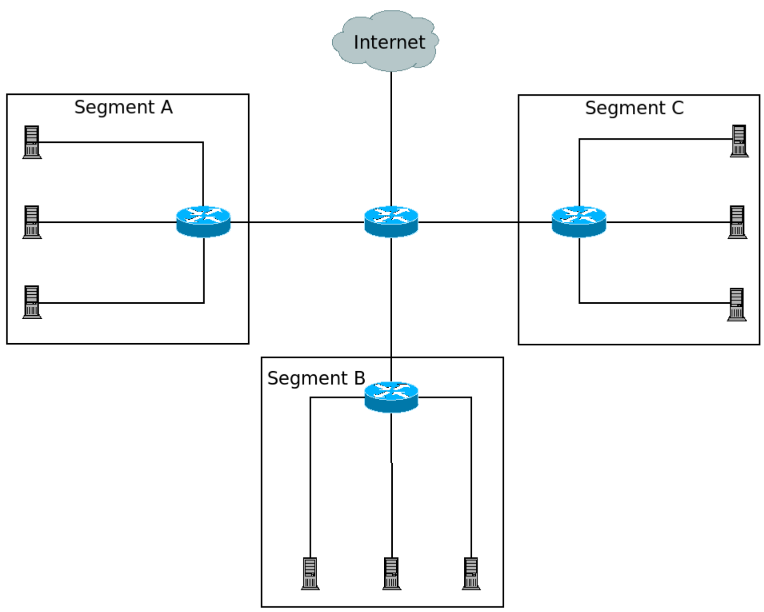



Figure 10: Base network design

To simplify our task we use a simplified network design based on the network shown in Figure 10. We add and remove data diodes between the different segments in the network to discuss how the changes affect the network traffic. If necessary, we replace our single router with one ingress router and one egress router to provide extra separation of the network traffic.

| Transmit\Receive | Segment A | Segment B | Segment C | Internet |
|---|---|---|---|---|
| **Segment A** | X | X | X | X |
| **Segment B** | X | X | X | X |
| **Segment C** | X | X | X | X |
| **Internet** | X | X | X | X |

Table 9: Baseline for our test network as shown in Figure 10

The network design in Figure 10 is illustrated with three network segments with three devices in each segment. This is sufficiently complex to answer our questions. The answers will be equally valid for a more complex network designs with more segments and more devices per segment. The network design of the different network segments are not relevant for our discussion unless they have one or more data diodes in the segment. Regarding the use of wireless access points in the different networks segments, there is nothing that prohibits that. The use of Wi-Fi or other wireless communication solutions on a network segment or machine that is protected by an uni- or bi-directional data diode opens up a new entry point into that machine or network segment and the point of using a data diode is to limit the possible entry points to the network segment or machine. We do therefore not recommend the use of Wi-Fi or other wireless communication systems on networks that is protected by an uni- or bi-directional data diode.

For each network design that we present and discuss we use the results from the default route test presented in Table 3 to answer the following questions:

- Which network segments are network segment X able to transmit packets to?
- Which network segments are network segment X able to receive packets from?
- Which machines in a network segment are unreachable from outside the network segment?
- Which machines in a network segment are unreachable from inside the network segment?

The last two questions are only applicable when a data diode is placed inside a network segment and are therefore only answered in these cases. To help us visualise the answer we populate Table 9 where the rows represent transmitting network segments and the columns represent receiving network segments. The X'es are placed where it is possible to transmit packets from the transmitter network segment to the receiver network segment.

Some of our network design have alternate designs with small changes from the primary design. These versions are shown in Appendix D.

One thing that is important to note is that we look at what probably would happen if we replace any network link with two uni-directional links with one data diode each, creating a bi-directional link that supports bi-directional protocols such as TCP. We are not discussing the effects of replacing a link with an uni-directional link using only one data diode. As noted in Section 4.2 a uni-directional data diode link needs to use a proxy that terminates the TCP session translating it to a protocol suitable for data diodes such as UDP and transmits it across the data diode where another proxy translates it back and starts a new session. The proxy solution limits the number of protocols that can be transmitted across the data diode, as the proxy software needs to support the protocols transmitted across the data diode.

### 8.1.1 Limitiations on network segmenting

It is important to remember that any compromised node in a network segment that has access to the Internet can attack any other network segment with access to the Internet by transmitting packets through a proxy located somewhere on the Internet. This bypasses security arrangements such as bi-directional data diodes and firewalls but gives the attacker no additional benefits over attacking the other segment directly through the proxy.

### 8.1.2 Design 1

For our first custom network design we replace the links from the core router to segment A and segment C with diode links while the link to segment B remains the same as before as shown in Figure 11. It should also be noted that this is basically the same network design we used to test the detection performance of an IDS.

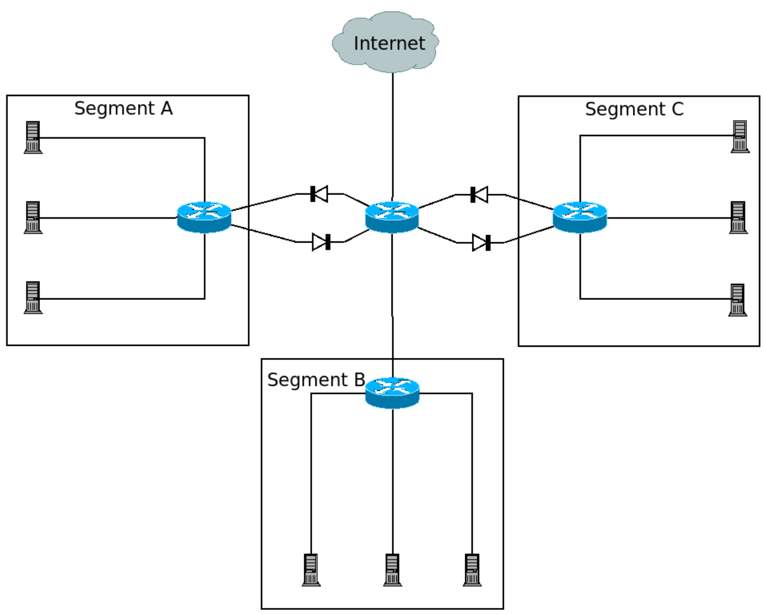



Figure 11: Design 1: Replacing normal links with data diode links

This network design allows for all three network segments to transmit and receive packets from each other as well as the Internet as shown in Table 10. This is the same as in our baseline design. This is expected as we replace one bi-directional link with two uni-directional links – one in each direction, without any other changes to the network design.

This design is more expensive and complex than our baseline design. One of the few things that we can argue is better for this design is that the network administrators have to manually enter

Table 10: Design 1: Replacing normal links with data diode links, as shown in Figure 11

| Transmit\Receive | Segment A | Segment B | Segment C | Internet |
|---|---|---|---|---|
| **Segment A** | X | X | X | X |
| **Segment B** | X | X | X | X |
| **Segment C** | X | X | X | X |
| **Internet** | X | X | X | X |

routing information at the endpoints of the diodes. Therefore the network administrator is forced to consider how routing should be applied in the network and can apply more limiting routing rules than only providing a default route, i.e. use variants of blackholing or no default route. One can also argue that manual configuration is a bad thing prone to misconfiguration.

We do therefore not recommend any network design based on this design.

#### 8.1.3 Design 2

For our second network design, shown in Figure 12, we split the core router from Figure 11 into two separate routers where one of them is dedicated to incoming traffic (ingress) and the other is dedicated to outgoing traffic (egress). Segments A and C are connected to both routers with the ingress and egress diode connected to the corresponding router, while segment B is connected to the egress router only through a traditional bi-directional link. There is also a diode that allows traffic to pass from the ingress to the egress router.

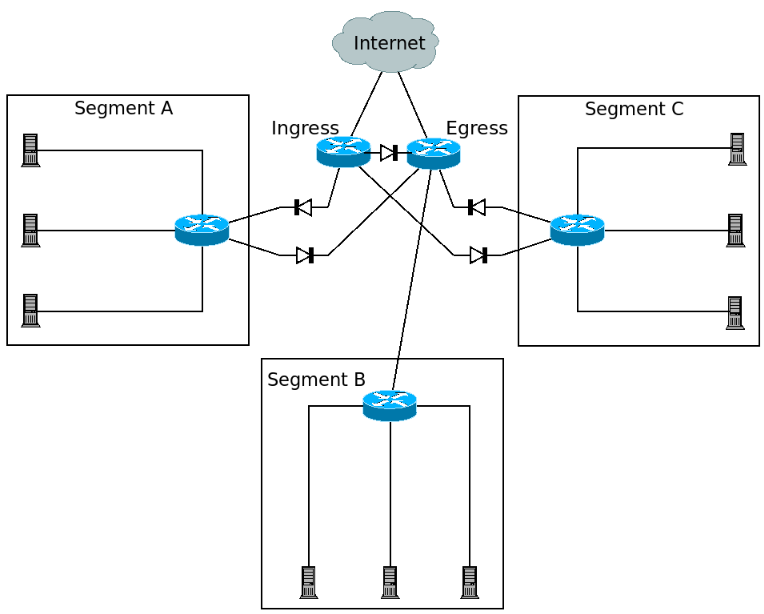



Figure 12: Design 2: Replacing the single middle router with separate ingress and egress routers

Table 11: Design 2: Second network design, data diodes and dual core routers shown in Figure 12

| Transmit\Receive | Segment A | Segment B | Segment C | Internet |
|---|---|---|---|---|
| **Segment A** | X | X | | X |
| **Segment B** | | X | | X |
| **Segment C** | | X | X | X |
| **Internet** | X | X | X | X |

Table 12: Design 2: Alternate version of the second network design, shown in Figure 17

| Transmit\Receive | Segment A | Segment B | Segment C | Internet |
|---|---|---|---|---|
| **Segment A** | X | | | X |
| **Segment B** | X | X | X | X |
| **Segment C** | | | X | X |
| **Internet** | X | X | X | X |

We have also created an alternative version of this design where we move the link to segment B from the egress router to the ingress router as shown in Figure 17 in Appendix D.

It is important to notice that we assume that the egress router routes traffic to segments behind diodes to the interfaces that they are connected to. For instance, if segment A tries to transmit a packet to segment C, the egress router is assumed to be configured to route that packet to the interface connected to segment C. This packet will not reach segment C, since that link is the receiving end of the data diode from segment C While this is not strictly blackholing, as we do not route traffic to nowhere, the effect is the same, since the traffic routed to certain interfaces cannot reach its destination.

Table 12 shows the connectivity for the main design and Table 11 shows the connectivity for the alternative design. In both cases, segments A and C cannot transmit packets to or receive packets from each other. The difference between the two designs is the connectivity for segment B. In the main design, segment B can receive packets from segment A and B, but cannot transmit packets to segments A and C. In the alternative design, segment B cannot receive packets from segment A and B, but can transmit packets to those two segments.

Both the main and alternative design will contain attacks within some of the segments, and is therefore recommended when there are two or more network segments that do not require any interconnectivity, while other segments are allowed to either transmit or receive data from any network segment.

### 8.1.4 Design 3

For our third design we are slightly simplifying design two. In this design segment B is connected by diodes while segment A is connected to the ingress router and segment C to the egress router as shown in Figure 13.

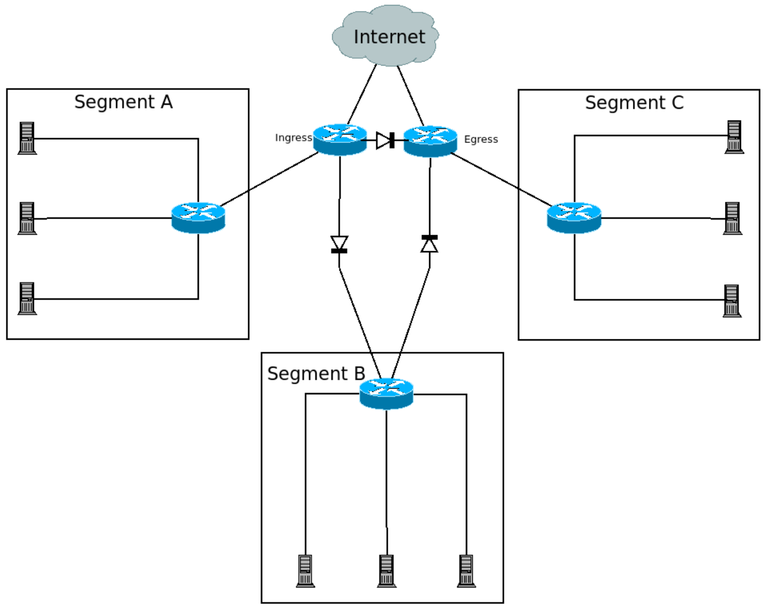



Figure 13: Design 3: Two normal connections and one bi-directional diode to two core routers

Table 13: Design 3: Two normal connections and one bi-directional diode to two core routers, as shown in Figure 13

| Transmit\Receive | Segment A | Segment B | Segment C | Internet |
|---|---|---|---|---|
| **Segment A** | X | X | X | X |
| **Segment B** | | X | X | X |
| **Segment C** | | | X | X |
| **Internet** | X | X | X | X |

This network design allows segment A to transmit packets to all the other network segments while it is unable to receive any packets from segment B and C. Segment C on the other hand can receive packets from all the other network segments but cannot transmit packets to segment A and B. Segment B can transmit packets to segment C and receive packets from segment A. All this is shown in Table 13.

This design prevents attackers that have breached segment B or C from reaching segment A, and is therefore recommended when there is at least one network segment that needs to be protected from attacks spreading from other network segments.

### 8.1.5 Design 4

For our fourth network design shown in Figure 14 we have placed bi-directional diodes on all of our segments while still using two separate core routers for egress and ingress.

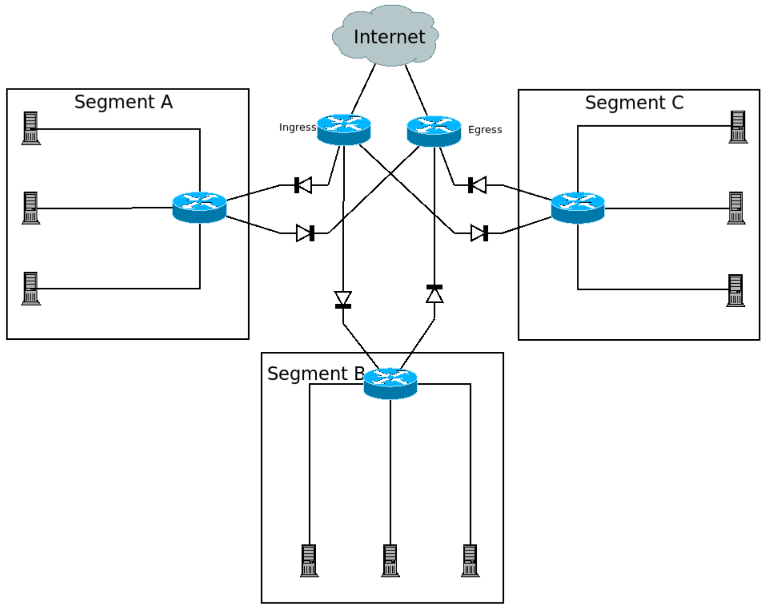



Figure 14: Design 4: All segments behind diodes

Table 14: Design 4: All segments behind diodes, as shown in Figure 14

| Transmit\Receive | Segment A | Segment B | Segment C | Internet |
|---|---|---|---|---|
| **Segment A** | X | | | X |
| **Segment B** | | X | | X |
| **Segment C** | | | X | X |
| **Internet** | X | X | X | X |

This network design does not allow any of the segments to transmit or receive packets from each other as shown in Table 14. The big question with this network design is why one would use this instead of three independent connections to the Internet as shown in Figure 18 in Appendix D? The only reason we could think of why someone would want to use this design is that they want to ensure that the different segments cannot transmit packets to each other. It is however possible to configure the router or a firewall in each segment to drop packets to the other segments and thereby achieve the same goal. When looking at the cost and complexity of configuring these two alternative solutions, the second alternative where each segment is directly connected to the Internet is simpler to configure and we have therefore assumed that it has fewer errors. The cost of our first alternative shown in Figure 14 is probably higher if we look at the initial cost as it needs more equipment then the second alternative. It might however be cheaper in the long term as we only have to pay for one connection instead of three separate connections.

We do not recommend this network design as it is more complex than having a separate connection to the internet which achieves the same level of protection of attacks spreading between the segments as this design.

### 8.1.6 Design 5

Our fifth design is an evolution of our fourth design where we have placed a bi-directional diode inside one of the network segments. In this case segment A, as show in Figure 15. This is the only network design where we show data diodes both inside the segments and between the different segments. For our last design we only show the internals of one segment because the interaction of the machines inside a network segment remains the same regardless of the segment interconnection.

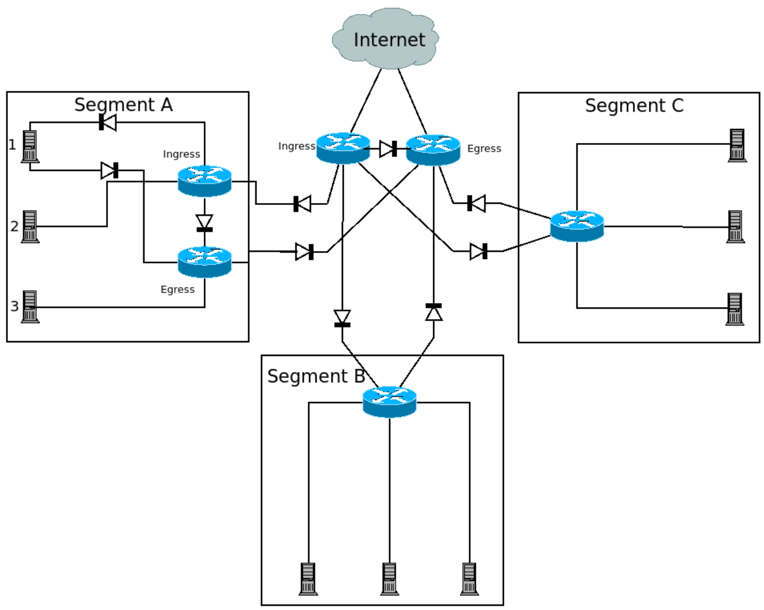



Figure 15: Design 5: All segments behind diodes and diodes inside the segments

Table 15: Design 5: All segments behind diodes and diodes inside the segments, as shown in Figure 15

| **Transmit \Receive** | **Machine 1** | **Machine 2** | **Machine 3** | **Internet** |
|---|---|---|---|---|
| **Machine 1** | X | | X | X |
| **Machine 2** | X | X | X | X |
| **Machine 3** | | | X | X |
| **Internet** | X | X | X | X |

For this network design we have chosen to focus on which machines in segment A that are reachable from inside and outside the network segment, since the inter segment connectivity is the same as for network design 4 shown in Table 14. We are using the same table with a couple of changes as seen here in Table 15. The biggest change is that segment A, B and C is replaced with machine 1, 2 and 3. The other small change is that Internet also means that the other segments can access that machine.

In this design machine 1 is unable to transmit packets to machine 3, while it can receive packets from machine 2. Machine 2 can transmit packets to both machine 1 and 2, but it can not receive packets from any of them. Machine 3, on the other hand, is able to receive packets from both machine 1 and 2, but unable to transmit packets to them. All of the machines can transmit and receive packets from any external segment and the Internet as shown in Table 15.

As this is basically a repetition of design 4 and 3 we do recommend the network design inside segment A for the same reason we recommend design 3. We do not recommend the design between the segments for the same reasons as in design 4.

### 8.1.7 Design 6

For our sixth design we have implemented a load-balancing or proxy scenario with data diodes. All the incoming traffic is routed through machine 1 that may either forward it to machine 2 or 3 or return with a response. Machine 2 or 3 then responds to the incoming traffic. This is our only design that only focuses on traffic inside a network segment as the results from our previous five designs are the same if applied to machines inside a network segment. We have therefore selected to not recreate them inside a network segment. It is also possible to use this kind of network design between different network segments.

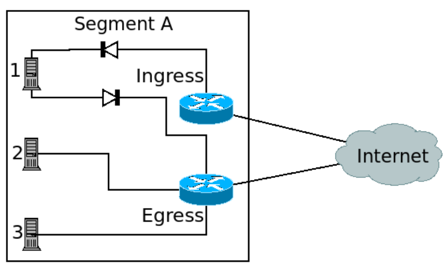



Figure 16: Design 6: Data diode in a load-balancing scenario

Table 16: Design 6: Data diode in a load-balancing scenario, as shown in Figure 16

| **Transmit \Receive** | **Machine 1** | **Machine 2** | **Machine 3** | **Internet** |
|---|---|---|---|---|
| **Machine 1** | X | X | X | X |
| **Machine 2** | | X | X | X |
| **Machine 3** | | X | X | X |
| **Internet** | X | ? | ? | X |

One unique feature in this design is that there is no direct connection between the ingress and egress router, thereby forcing all the traffic through machine 1, regardless of if it is the final destination of the traffic or not.

Table 16 shows the possible connection in this network segment. Machine 1 is able to transmit packets to machines 2 and 3 but unable to receive any packets from machines 2 and 3. Machines 2 and 3 can receive packets from machine 1 as well as each other, they are however only able to transmit packets between themselves and out to the Internet. The Internet can only directly transmit packets to machine 1, it should however be able to indirectly transmit packets to machines 2 and 3 which therefore have gotten a ? symbol as that depends on the configuration of machine 1. All the machines are however able to transmit packets out on the Internet.

We recommend this design for users that uses a proxy or a load-balancer that does not need to communicate with the other servers that it functions as a proxy or load-balancer for.

### 8.1.8 Outside interaction and classical prevention methods

This section answers the third research question by comparing the network designs using bi-directional data diodes with classical prevention methods.

We previously mentioned that a machine that is connected to the Internet is vulnerable to attacks from the Internet. The question that we need to answer is if it is easier to attack a machine inside a network from the inside that network than from outside that network? For instance is it easier to attack segment C from segment A than to attack segment C from the Internet? For the use cases that we have described where no additional protection such as firewalls, ACL, Network Address Translation (NAT) etc. the answer is no. It is as easy to attack any segment or machine from the Internet as it is from inside the network. We know that NAT is not designed to work as a security measure even if it prevents attackers direct access to all the machines in a network behind a router or modem using NAT

However, any data center or company network and even most home networks have at least one firewall present in the network. In addition, many Internet Service Providers (ISPs) blocks certain ports that are often abused by hackers. Thereby making it generally easier to attack a network from the inside than from the Internet.

Because of this, attackers often attack less secured devices in the network to gain an entry point into the network or network segment, which they then use to attack other machines. Malware might also work in a similar manner. Our goals is therefore to minimize the attack surface from within our network and more specifically the attack surface between network segments. We have shown in designs 2 to 5 that bi-directional data diodes combined with separate ingress and egress routers limits the access between network segments, thereby limiting the attack surface between the network segments.

#### Routing tables

We start by discussing routing tables, since that is something that we have modified in our tests. As shown in Table 3 in Chapter 7 it is possible to prohibit communication from one network to another with changes to the routing table such as blackholing. And thereby limit the reach of an attacker or malware in the network.

The reason to use data diodes over blackholing or other changes to the routing table is that a data diode guarantees that communications can only travel in one direction regardless of what the user does. More benefits and drawbacks to each solution is mentioned later in this subsection.

#### Firewalls and ACL

We have chosen to group firewalls and ACL together as they primarily work the same way. While ACLs are often included in routers and switches, firewalls are often separate devices monitoring the network traffic. Both use packet inspection to decide if a packet shall be forwarded, dropped or rejected. This allows the user to block parts of the traffic between segments while it still allows the rest of it. Both firewalls and ACLs can be configured to drop or block all traffic from one segment to another.

Stateful firewalls and firewalls that perform deep packet inspection works slightly different than ACLs as they can decide to reject, drop or allow packets based on the packet content or current active connections through the firewall.

As with the routing tables, the reason to use data diodes is that they ensure that communication can only travel in one direction regardless of what the user does. There is, however, no reason to not use a firewall or ACL together with a data diode to provide more layers of protection. The data diode guarantees the uni-directional network link. While the firewall or ACL can forward only the traffic that is needed by the services behind the data diode

Placing a stateful firewall on one of the links creating a bi-directional data diode link does not make sense. It needs to monitor the traffic flowing both ways to detect when sessions are established. It is therefore recommended to place stateful firewalls on a normal bi-directional link.

#### Pros and Cons

Since most of the benefits and drawbacks are the same for firewalls, ACLs, and routing tables when compared with data diodes, we have selected to present one common list instead of two very similar lists.

Firewalls and ACLs one benefit over both routing tables and data diodes. That is that they can block or pass through only certain parts of the network traffic. Except for this, they have the same benefits and drawbacks over a data diode as routing table changes.

**Pros data diodes**

- Unhackable
- Reduced attack surface
- Software misconfiguration can not enable bi-directional comunication across the data diode

**Cons data diodes**

- Expensive
- More complex configuration

**Pros traditional solutions**

- Cheaper
- Simpler to configure

**Cons traditional solutions**

- Can be hacked
- Larger attack surface
- Misconfiguration can remove the assumed security of the configuration

It should be noted that anyone with physical access can bypass any security mechanisms by changing the physical connection such as directly connecting two networks together instead of running it through a data diode, or a firewall etc.

So should one use traditional solutions or bi-directional data diodes? There is no simple answer to this question as it all depends on what is most important for the users of the network: security, money and/or ease of configuration.

Using data diodes provides better security than using the traditional methods mentioned in this chapter. A cost/benefit analysis should be performed to evaluate if the improved security from using data diodes is worth the cost.

The best solution from a security standpoint is to combine the traditional methods with bi-directional data diodes to create multiple layers of security.

## 8.2 IDS performance

The metrics we have used to measure IDS performance are detection rate and total packet processing time as shown in Section 4.5.1 and Chapter 7. We first evaluate and discuss the results relevant to hypothesis 1, where we look at the detection performance. Then we move on to evaluate and discuss the results for hypothesis 2 where we look at the resource usage.

### 8.2.1 Detection performance

The tests for hypothesis 1 assume that the IDS when placed in Figure 6 shall generate alarms for all UDP traffic between nR1 and nR2. The Snort rules for nR1 is shown in line 1 of Listing 1 and explained in Section 4.5. Similar rules are used for nR2. The second line of the rule was used to verify our results as it generates an alert for the rest of the UDP packets.

```
1 alert udp [10.0.3.0/24] any -> any any (msg: "Connection from: 10.0.3.0/24 detected"; sid:1;)
2 #alert udp ![10.0.3.0/24] any -> any any (msg: "Normal traffic detected"; sid:2;)
```

Listing 10: Snort rule for nR1

The expected detection performance result for a rule like this in our setting where we only generate UDP packets are a TPR of 100 %, a FPR of 0 % as well as a TNR of 100 % and a FNR of 0. Since we do not have any TCP/IP sources of traffic or edge cases where some packets from nR1 or nR2 should be allowed to be transmitted between the networks, which then might results in a few false positive detections or false negative detections.

The results shown in Chapter 7 indicates TPR of 100 %, a FPR of 0 % as well as a TNR of 100 % and a FNR of 0 % in both networks. Since Snort generates the same amount of alerts as packets from nR1 being received at nR2 and the other way around. This was verified by comparing the output from a `tcpdump` command that gave us an output of all the packets in the pcap file with the output of Snort where line two in Listing 10 generates an alert for all detected UDP packets.

We were unable to get Snort to drop any packets in our test. It is, nevertheless, interesting to look at the impact dropped packets might have on our detection results. This can be done by first calculating the expected dropped packet rate and then calculate an estimated FNR. This can be done as long as we know the *Snort packet processing rate*, the *packet rate* of the monitored link, the *total number of attack packets* and the *total number of packets*. The calculations necessary to do this is shown down bellow:

$$\text{dropped packet rate} = \text{packet rate} - \text{Snort packet processing rate}$$

$$\text{false negative rate} \approx \text{dropped packet rate} * \frac{\text{total number of attack packets}}{\text{total number of packets}}$$

To estimate the FNR of our tests do we have the following numbers from our tests:

- packet rate = 600 Pkts/second
- Snort packet processing rate = 105906 Pkts/second (taken from Listing 9)

Which gives us a dropped packet rate of 0:

$$600 - 105906 = 0$$

The calculations gives us a negative number, any number bellow 0 is regarded as 0 as it is impossible to have less than 0 dropped packets.

The estimated FNR is then 0:

$$0 * \frac{200000}{600004} = 0$$

There is therefore no change to the detection performance due to any potential dropped packets in our tests.

How important dropped packets are when looking at detection performance can be discussed, as it is possible to mitigate the dropped packets either by increasing the power of the system that runs Snort or by splitting the workload out across multiple computers. This increases the cost and the complexity of the IDS.

Based on our results we conclude that **Hypothesis 1** is true, since we have shown that the detection performance and number of packets analysed is consistent between our two networks when we uses Snort to monitor a network segment behind data diodes.

### 8.2.2 IDS resource usage

Regarding **hypothesis 2:** where we look at IDS detection performance when we compare IDS monitoring a data diode link compared to an IDS monitoring a traditional link our results are supportive of our hypothesis. While the number of alerts and transmitted/received packets is the same for both the traditional and data diode networks, the total number of analysed packets for the data diode network are halved compared to the traditional network, if we disregard the four ARP requests and responses that are a part of the traditional network but not a part of the data diode networks. The difference between the total number of packets are going to be smaller and larger in a real-life scenario where one for instance requests some information from the server instead of ping which we used. The reason for this is that many responses to a request contains more data than the request itself and either needs to use more or larger packets to transmit the data back to the user, thereby leaving the IDS with a fraction of the traffic to monitor. While this might look like the biggest advantage for monitoring data diode links it is at also its greatest weakness, since it only allows the IDS to monitor traffic either entering or leaving the network, not both at the same time. This however depends on the rules and configuration of the IDS. If the goal is to only detect incoming attacks or someone receiving data, they should then be fine with one IDS monitoring the corresponding link.

In addition to processing fewer packets there is a difference in packet processing time between our two tests where the IDSes monitoring a link in a bi-directional data diode processes packets for approximately two seconds less than the IDSes monitoring a traditional link. While this is no direct measurement of computer resource usage of an IDS is it an indicator together with the packet processing rate of how powerful the system running the IDS needs to be to not bottleneck the connection.

Based on our results we conclude that **Hypothesis 2** is true, as we there is no difference in the detection performance, while the packet processing time and thereby the computer resource usage is lower for the IDS monitoring a link in a bi-directional data diode connection.

# 9 Conclusion

This thesis introduces a novel use of data diodes to create what we call a bi-directional data diode. A bi-directional data diode replaces one traditional bi-directional Ethernet link with two data diodes. The data diodes are connected in opposite directions. This configuration provides bi-directional traffic across the bi-directional data diode while guaranteeing uni-directional traffic across each data diode.

## 9.1 Network design

We have shown that introducing a single bi-directional data diode does not improve security or help to contain an attack.

Security is improved by enforcing segmentation of a network if a bi-directional data diodes uni-directional links are connected to separate ingress and egress routers. This segmentation will limit the reach of an attacker that has successfully attacked parts of the network.

A number of network designs using multiple bi-directional data diodes have been proposed. Most of these designs prevent access between at least two networks segments or machines.

## 9.2 IDS performance

The use of bi-directional data diodes have little or no impact on IDS detection performance. We have shown that the TPR, FPR, TNR and FNR is the same regardless of an IDS being placed in a network with bi-directional data diodes or without bi-directional data diodes. Using an IDS to monitor a single link in a bi-directional data diode might decrease the resource usage as the number of packets the IDS needs to process is lower than for the same link in a traditional network.

## 9.3 Future works

The following future works are suggested:

- Rerun our experiments with TCP/IP packets instead of UDP packets
- Rerun our experiments with physical hardware
- Rerun our tests in a newer version of ns-3 when full-duplex CSMA links are implemented [35]

# 10 Acronyms and Definitions

## 10.1 Acronyms

**ARP** Address Resolution Protocol

**ACL** Access-Control List

**CSMA** Carrier-sense multiple access

**ISP** Internet Service Provider

**MAC** Media Access Control

**NAT** Network Address Translation

**NIC** Network Interface Controller

**IDS** Intrusion Detection System

**IP** Internet Protocol

**IPS** Intrusion Prevention System

**OS** Operating System

**TCP/IP** Transmission Control Protocol/Internet Protocol

**TCP** Transmission Control Protocol

**UDP** User Datagram Protocol

**UTP** Unshielded Twisted Pair

**FPR** False Positive Rate

**TPR** True Positive Rate

**FNR** False Negative Rate

**TNR** True Negative Rate

**VM** Virtual Machine

## 10.2 Definitions

**Data diode** A network link that has been modified to send traffic only in one direction thereby creating a uni-directional link.

**Bi-directional Data diode** A bi-directional data diode replaces one traditional bi-directional Ethernet link with two data diodes. The data diodes are connected in opposite directions.

**True positive** A test result that correctly indicates the presence of a condition or characteristic [36].

**False positive** A test result which wrongly indicates that a particular condition or attribute is present [37].

**True negative** A test result that correctly indicates the absence of a condition or characteristic [38].

**False negative** A test result which wrongly indicates that a particular condition or attribute is absent [39].

# Bibliography

[1] Stevens, M. W., Science, D., (Australia), T. O., Electronics, & (Australia), S. R. L. 1999. *An Implementation of an optical data diode / Malcolm W. Stevens*. DSTO Electronics and Surveillance Research Laboratory Salisbury, S. Aust. URL: http://www.dsto.defence.gov.au/corporate/reports/DSTO-TR-0785.pdf.

[2] Heo, Y. & Na, J. Oct 2016. Development of unidirectional security gateway appliance using intel 82580eb nic interface. In *2016 International Conference on Information and Communication Technology Convergence (ICTC)*, 1194–1196. doi:10.1109/ICTC.2016.7763404.

[3] Kim, J. & Na, J. Feb 2017. A study on one-way communication using pf_ring zc. In *2017 19th International Conference on Advanced Communication Technology (ICACT)*, 301–304. doi:10.23919/ICACT.2017.7890102.

[4] Okhravi, H., Sheldon, F. T., & Haines, J. *Data Diodes in Support of Trustworthy Cyber Infrastructure and Net-Centric Cyber Decision Support*, 203–216. Springer, 2013. URL: http://dx.doi.org/10.1007/978-3-642-38134-8_10, doi:10.1007/978-3-642-38134-8_10.

[5] Yun, J.-H., Chang, Y., Kim, K.-H., & Kim, W. 2017. Security validation for data diode with reverse channel. In *Critical Information Infrastructures Security*, 271–282. Springer. doi:10.1007/978-3-319-71368-7_23.

[6] Greenberg, A. 2018. The untold story of notpetya, the most devastating cyberattack in history. *Wired, August*.

[7] Lee, M., Mercer, W., Rascagneres, P., & Williams., C. Player 3 has entered the game: Say hello to 'wannacry'. https://blog.talosintelligence.com/2017/05/wannacry.html. Visited 12.12.18.

[8] Issariyakul, T. & Hossain, E. 2012. Introduction to network simulator 2 (ns2). In *Introduction to Network Simulator NS2*, 21–40. Springer.

[9] Chen, H., Clark, J. A., Shaikh, S. A., Chivers, H., & Nobles, P. Feb 2010. Optimising ids sensor placement. In *2010 International Conference on Availability, Reliability and Security*, 315–320. doi:10.1109/ARES.2010.92.

[10] Aryachandra, A. A., Arif, Y. F., & Anggis, S. N. May 2016. Intrusion detection system (ids) server placement analysis in cloud computing. In *2016 4th International Conference on Information and Communication Technology (ICoICT)*, 1–5. doi:10.1109/ICoICT.2016.7571954.

[11] Proxmox. Proxmox wiki main page. https://pve.proxmox.com/wiki/Main_Page. Visited 12.12.18.

[12] Leedy, P. D. 2015. Practical research : planning and design.

[13] Yin, R. 2017. *Case Study Research and Applications: Design and Methods*. SAGE Publications. URL: https://books.google.no/books?id=6DwmDwAAQBAJ.

[14] Kehe, W., Fei, C., & Wenchao, C. 2009. The technique of network diode. In *2009 First International Conference on Information Science and Engineering*.

[15] de Freitas, M. B., Rosa, L., Cruz, T., & Simões, P. 2019. Sdn-enabled virtual data diode. In *Computer Security*, Katsikas, S. K., Cuppens, F., Cuppens, N., Lambrinoudakis, C., Antón, A., Gritzalis, S., Mylopoulos, J., & Kalloniatis, C., eds, 102–118, Cham. Springer International Publishing.

[16] Hofstadter, D. R. 1979. *Gödel, Escher, Bach: An Eternal Golden Braid*. New York: Basic Books.

[17] Rieback, M. R., Crispo, B., & Tanenbaum, A. S. 2006. Is your cat infected with a computer virus? In *Pervasive Computing and Communications, 2006. PerCom 2006. Fourth Annual IEEE International Conference on*, 10–pp. IEEE.

[18] Barker, R. T. & Cheese, C. J. Oct 2012. The application of data diodes for securely connecting nuclear power plant safety systems to the corporate it network. In *7th IET International Conference on System Safety, incorporating the Cyber Security Conference 2012*, 1–6. doi:10.1049/cp.2012.1514.

[19] U.S. Department Homeland of Security and NCCIC. Seven steps to effectively defend industrial control systems. https://ics-cert.us-cert.gov/sites/default/files/documents/Seven%20Steps%20to%20Effectively%20Defend%20Industrial%20Control%20Systems_S508C.pdf. Visited 30.05.19.

[20] Schlicher, B. G., MacIntyre, L. P., & Abercrombie, R. K. Jan 2016. Towards reducing the data exfiltration surface for the insider threat. In *2016 49th Hawaii International Conference on System Sciences (HICSS)*, 2749–2758. doi:10.1109/HICSS.2016.345.

[21] Fibersystem AB. Data diode bidirectional 1 gbit tempest level a/emsec/rös u1 datasheet. https://www.fibersystem.com/wp-content/uploads/FS19189-Datasheet-Data-Diode-Bidirectional-1-Gbit-Tempest-R1.pdf. Visited 23.5.19.

[22] Owl cyber defense. Immediate release owl cyber defense unveils “recon” bidirectional data diode-based cybersecurity solution. http://library.owlcyberdefense.com/press_2018-may-31/page/1, May 2018. Visited 23.5.19.

[23] diode. https://en.oxforddictionaries.com/definition/diode. Visited 02.05.19.

[24] Fibersystem AB. Data diode middleware datasheet. https://www.fibersystem.com/wp-content/uploads/FS17101-Datadiode-Middleware-DDMW-Datasheet-1.pdf. Visited 22.5.19.

[25] IEEE 802.3 Working Group. Aug 2018. Ieee standard for ethernet. *IEEE Std 802.3-2018 (Revision of IEEE Std 802.3-2015)*, 1–5600. doi:10.1109/IEEESTD.2018.8457469.

[26] Lc-sc-fiber-connectors. https://commons.wikimedia.org/wiki/File:Lc-sc-fiber-connectors.jpg. Visited 23.5.19.

[27] Mike1024. Close-up photo of an uncrimped, transparent rj-45 plug. https://en.wikipedia.org/wiki/Modular_connector#/media/File:Uncrimped_rj-45_connector_close-up.jpg. Visited 23.5.19.

[28] Snort community. Snort - Network Intrusion Detection & Prevention System. https://www.snort.org. Visited 13.12.18.

[29] øyvind Aasen. Using network diodes to contain malicious network traffic. 2018.

[30] nsam. What is ns-3. https://www.nsnam.org/about/. Visited 10.04.19.

[31] nsam. The difference between ns-2 and ns-3. https://www.nsnam.org/support/faq/ns2-ns3/. Visited 10.04.19.

[32] Open Information Security Foundation. suricata open source ids / ips / nsm engine. https://suricata-ids.org/. Visited 30.05.19.

[33] ns3 wiki. Howto make ns-3 interact with the real world. https://www.nsnam.org/wiki/HOWTO_make_ns-3_interact_with_the_real_world. Visited 01.06.19.

[34] ns3 wiki. Howto use linux containers to set up virtual networks. https://www.nsnam.org/wiki/HOWTO_Use_Linux_Containers_to_set_up_virtual_networks. Visited 01.06.19.

[35] Henderson, T. Bug 2354 - code review: full duplex csma. https://www.nsnam.org/bugzilla/show_bug.cgi?id=2354. Visited 02.05.19.

[36] true positive. https://en.oxforddictionaries.com/definition/true_positive. Visited 29.05.19.

[37] false positive. https://en.oxforddictionaries.com/definition/false_positive. Visited 29.05.19.

[38] true negative. https://en.oxforddictionaries.com/definition/true_positive. Visited 29.05.19.

[39] false negative. https://en.oxforddictionaries.com/definition/false_negative. Visited 29.05.19.

# A ns-3 output

This is the results of test 1 used to populate Table 3 in Section 7.1.

```
At time 1s client sent 1024 bytes to 10.0.2.2 port 9
At time 1.00101s server received 1024 bytes from 10.0.1.2 port 49153
At time 1.00101s server sent 1024 bytes to 10.0.1.2 port 49153
At time 1.01202s client received 1024 bytes from 10.0.2.2 port 9
At time 2s client sent 1024 bytes to 10.0.3.2 port 9
At time 2.00101s server received 1024 bytes from 10.0.1.2 port 49154
At time 2.00101s server sent 1024 bytes to 10.0.1.2 port 49154
At time 2.00602s client received 1024 bytes from 10.0.3.2 port 9
At time 3s client sent 1024 bytes to 10.0.3.2 port 9
At time 3.00001s server received 1024 bytes from 10.0.2.2 port 49153
At time 3.00001s server sent 1024 bytes to 10.0.2.2 port 49153
At time 3.00002s client received 1024 bytes from 10.0.3.2 port 9
At time 4s client sent 1024 bytes to 10.0.2.2 port 9
At time 4.00001s server received 1024 bytes from 10.0.3.2 port 49153
At time 4.00001s server sent 1024 bytes to 10.0.3.2 port 49153
At time 4.00002s client received 1024 bytes from 10.0.2.2 port 9
```

Listing 11: ns-3 default route output

```
At time 1s client sent 1024 bytes to 10.0.2.2 port 9
At time 1.00101s server received 1024 bytes from 10.0.1.2 port 49153
At time 1.00101s server sent 1024 bytes to 10.0.1.2 port 49153
At time 1.01202s client received 1024 bytes from 10.0.2.2 port 9
At time 2s client sent 1024 bytes to 10.0.3.2 port 9
At time 2.00101s server received 1024 bytes from 10.0.1.2 port 49154
At time 2.00101s server sent 1024 bytes to 10.0.1.2 port 49154
At time 2.00602s client received 1024 bytes from 10.0.3.2 port 9
At time 3s client sent 1024 bytes to 10.0.3.2 port 9
At time 4s client sent 1024 bytes to 10.0.2.2 port 9
```

Listing 12: ns-3 no default route output

```
At time 1s client sent 1024 bytes to 10.0.2.2 port 9
At time 1.00101s server received 1024 bytes from 10.0.1.2 port 49153
At time 1.00101s server sent 1024 bytes to 10.0.1.2 port 49153
At time 1.01202s client received 1024 bytes from 10.0.2.2 port 9
At time 2s client sent 1024 bytes to 10.0.3.2 port 9
At time 2.00101s server received 1024 bytes from 10.0.1.2 port 49154
At time 2.00101s server sent 1024 bytes to 10.0.1.2 port 49154
At time 2.00602s client received 1024 bytes from 10.0.3.2 port 9
At time 3s client sent 1024 bytes to 10.0.3.2 port 9
At time 4s client sent 1024 bytes to 10.0.2.2 port 9
At time 4.00001s server received 1024 bytes from 10.0.3.2 port 49153
At time 4.00001s server sent 1024 bytes to 10.0.3.2 port 49153
```

Listing 13: ns-3 blackhole nR2 on nR1d output

```
At time 1s client sent 1024 bytes to 10.0.2.2 port 9
At time 1.00101s server received 1024 bytes from 10.0.1.2 port 49153
At time 1.00101s server sent 1024 bytes to 10.0.1.2 port 49153
At time 1.01202s client received 1024 bytes from 10.0.2.2 port 9
At time 2s client sent 1024 bytes to 10.0.3.2 port 9
At time 2.00101s server received 1024 bytes from 10.0.1.2 port 49154
At time 2.00101s server sent 1024 bytes to 10.0.1.2 port 49154
At time 2.00602s client received 1024 bytes from 10.0.3.2 port 9
At time 3s client sent 1024 bytes to 10.0.3.2 port 9
At time 3.00001s server received 1024 bytes from 10.0.2.2 port 49153
At time 3.00001s server sent 1024 bytes to 10.0.2.2 port 49153
At time 4s client sent 1024 bytes to 10.0.2.2 port 9
```

Listing 14: ns-3 blackhole nR1 on nR2d output

# B Code

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * This program is free software; you can redistribute it and/or
   modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA
   02111-1307  USA
 */


/* Overview of the network
 *
 *              nT
 *          -<-  |  -<-
 *         /   \ | /   \
 * nR1-nR1d     nTd     nR2d-nR2
 *         \   /   \   /
 *          ->-     ->-
 */

#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/csma-module.h"
#include "ns3/arp-cache.h"
#include "ns3/ipv4-static-routing-helper.h"
#include "ns3/netanim-module.h"
#include "ns3/flow-monitor-helper.h"
```

```
// Normal C++ stuff
#include <iostream>
#include <fstream>
#include <string>

using namespace ns3;
using std::cout;

NS_LOG_COMPONENT_DEFINE ("NetworkDiodeTest");


// A function to create a data diode connection
void
CreateDiode (Ptr<Node> sender,
    Ptr<Node> receiver,
    char const* adress,
    char const* subnetMask,
    char const* baseAdr,
    Ipv4StaticRoutingHelper* ipv4RoutingHelper,
    Ipv4Address destAdr,
    Ipv4Mask destMask,
    bool pcap=false
    )
{
  // Create the network link
  CsmaHelper csma;

  NodeContainer nodes = NodeContainer (sender, receiver);

  // Create the "network interfaces" and add them to the appropriate nodes
  NetDeviceContainer diodes;
  diodes = csma.Install(nodes);

  // Configure the interfaces as data diodes
  Ptr<CsmaNetDevice> diodeS = DynamicCast<CsmaNetDevice> (diodes.Get(0));
  diodeS->SetReceiveEnable (false);
  diodeS->SetSendEnable (true);

  Ptr<CsmaNetDevice> diodeR = DynamicCast<CsmaNetDevice> (diodes.Get(1));
  diodeR->SetReceiveEnable (true);
  diodeR->SetSendEnable (false);
```

```

// Initalize Internet if needed
InternetStackHelper stack;
// Ensure that the node does not already have an initialized IP
    stack
if (sender->GetObject<Ipv4> () != 0)
{
  NS_LOG_INFO ("Sender already has an initialized IP stack");
}
else
{
  stack.Install(sender);
}
if (receiver->GetObject<Ipv4> () != 0)
{
  NS_LOG_INFO ("Receiver already has an initialized IP stack");
}
else
{
  stack.Install(receiver);
}

// Set IP addresses
Ipv4AddressHelper address;
address.SetBase (adress, subnetMask, baseAdr);
Ipv4InterfaceContainer interfacesDiodes;
interfacesDiodes = address.Assign (diodes);

// Variables used for static routing
Ipv4Address destIntAdress = interfacesDiodes.GetAddress(1);
uint32_t numInterface = diodeS->GetIfIndex();

Ptr<Ipv4> ipv4S = sender->GetObject<Ipv4> ();

// Use static routing for the diodes, hopefully allowing for "
    loops"
Ptr<Ipv4StaticRouting> staticRouteT = ipv4RoutingHelper->
    GetStaticRouting (ipv4S);
staticRouteT->AddNetworkRouteTo (destAdr, destMask, destIntAdress,
     numInterface);
// Comment out to disable default routes
staticRouteT->SetDefaultRoute (destIntAdress, numInterface);

// Manually fill the ARP cache of the transmit node
Ptr<ArpCache> arpT = CreateObject<ArpCache> ();
arpT->SetAliveTimeout (Seconds (3600 * 24)); // Keep the ARP table
```

```
       entry for one day...

  ArpCache::Entry * entry = arpT->Add (interfacesDiodes.GetAddress(1));
  entry->SetMacAddress(Mac48Address::ConvertFrom(diodeR->GetAddress()));
  entry->MarkPermanent ();

  // Add the cache to the transmit node
  std::pair<Ptr<Ipv4>, uint32_t> returnValue = interfacesDiodes.Get(0);
  Ptr<Ipv4> ipv4 = returnValue.first;
  uint32_t index = returnValue.second;
  Ptr<Ipv4Interface> diodeT = ipv4->GetObject<Ipv4L3Protocol> ()->GetInterface (index);
  arpT->SetDevice(diodeS, diodeT);
  diodeT->SetAttribute("ArpCache", PointerValue(arpT));

  // Enable packet trace on diode interfaces
  if (pcap)
  {
    csma.EnablePcap ("diode-send", diodeS, true);
    csma.EnablePcap ("diode-receive", diodeR, true);
  }
}

int
main (int argc, char *argv[])
{
  uint32_t nPackets = 100;
  uint32_t nClients = 10;

  CommandLine cmd;
  cmd.AddValue("nPackets", "Number of packets to echo", nPackets);
  cmd.AddValue("nClients", "Number of extra clients to run", nClients);
  cmd.Parse (argc, argv);

  Time::SetResolution (Time::NS);
  LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
  LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);

  // Create the nodes
  Ptr<Node> nT = CreateObject<Node> (); // node tramsittor
  Ptr<Node> nT2 = CreateObject<Node> (); // node tramsittor
  Ptr<Node> nR1 = CreateObject<Node> (); // first node receiver
```

```
Ptr<Node> nR2 = CreateObject<Node> (); // second node receiver
Ptr<Node> nTd = CreateObject<Node> (); // "transmitting" diode
Ptr<Node> nR1d = CreateObject<Node> (); // first "receiving" diode
Ptr<Node> nR2d = CreateObject<Node> (); // second "receiving"
    diode

Names::Add("nT", nT);
Names::Add("nT2", nT2);
Names::Add("nR1", nR1);
Names::Add("nR2", nR2);
Names::Add("nTd", nTd);
Names::Add("nR1d", nR1d);
Names::Add("nR2d", nR2d);

NodeContainer nodesT = NodeContainer (nTd, nT);
NodeContainer nodesR1 = NodeContainer (nR1d, nR1);
NodeContainer nodesR2 = NodeContainer (nR2d, nR2);

// Create the network link
CsmaHelper csma;

// Create the "network interfaces" and add them to the appropriate
    nodes
NetDeviceContainer sendNet;
sendNet = csma.Install(nodesT);
NetDeviceContainer reciveNet1;
reciveNet1 = csma.Install(nodesR1);
NetDeviceContainer reciveNet2;
reciveNet2 = csma.Install(nodesR2);

InternetStackHelper stack;
stack.Install(nodesT);
stack.Install(nodesR1);
stack.Install(nodesR2);

// Assign fixed IP-adresses to the networks
Ipv4AddressHelper address;
// Transmitt network
address.SetBase ("10.0.1.0" , "255.255.255.0");
Ipv4InterfaceContainer interfaceT;
interfaceT = address.Assign (sendNet);

// Receive network
address.SetBase ("10.0.2.0" , "255.255.255.0");
Ipv4InterfaceContainer interfaceR1;
interfaceR1 = address.Assign (reciveNet1);
```

```
// Second Receive network
address.SetBase ("10.0.3.0" , "255.255.255.0");
Ipv4InterfaceContainer interfaceR2;
interfaceR2 = address.Assign (reciveNet2);

// Enable static routing
Ipv4StaticRoutingHelper ipv4RoutingHelper;

// Configure nTd, nR1d, and nR2d to function as diodes
CreateDiode(nTd, nR1d, "192.168.0.0", "255.255.255.0", "0.0.0.1",
    &ipv4RoutingHelper, "10.0.2.0", "/24", true);
CreateDiode(nR1d, nTd, "192.168.0.0", "255.255.255.0", "0.0.0.3",
    &ipv4RoutingHelper, "10.0.1.0", "/24", true);
CreateDiode(nTd, nR2d, "192.168.0.0", "255.255.255.0", "0.0.0.5",
    &ipv4RoutingHelper, "10.0.3.0", "/24", true);
CreateDiode(nR2d, nTd, "192.168.0.0", "255.255.255.0", "0.0.0.7",
    &ipv4RoutingHelper, "10.0.1.0", "/24", true);

// Static routing
Ptr<Ipv4> ipv4nT = nT->GetObject<Ipv4> ();
Ptr<Ipv4> ipv4nR1 = nR1->GetObject<Ipv4> ();
Ptr<Ipv4> ipv4nR2 = nR2->GetObject<Ipv4> ();
Ptr<Ipv4> ipv4nTd = nTd->GetObject<Ipv4> ();
Ptr<Ipv4> ipv4nR1d = nR1d->GetObject<Ipv4> ();
Ptr<Ipv4> ipv4nR2d = nR2d->GetObject<Ipv4> ();

// Use static routing for the diodes hopefully allowing for "loops
    "
// The primary goal is to populate the adress of the other side of
    the diode
Ptr<Ipv4StaticRouting> staticRoutenT = ipv4RoutingHelper.
    GetStaticRouting (ipv4nT);
staticRoutenT->AddNetworkRouteTo (Ipv4Address ("10.0.2.0"),
    Ipv4Mask ("/24"), Ipv4Address ("10.0.1.1"), 1);
staticRoutenT->AddNetworkRouteTo (Ipv4Address ("10.0.3.0"),
    Ipv4Mask ("/24"), Ipv4Address ("10.0.1.1"), 1);

Ptr<Ipv4StaticRouting> staticRoutenR1 = ipv4RoutingHelper.
    GetStaticRouting (ipv4nR1);
staticRoutenR1->AddNetworkRouteTo (Ipv4Address ("10.0.1.0"),
    Ipv4Mask ("/24"), Ipv4Address("10.0.2.1"), 1);
staticRoutenR1->SetDefaultRoute (Ipv4Address("10.0.2.1"), 1);

Ptr<Ipv4StaticRouting> staticRoutenR2 = ipv4RoutingHelper.
    GetStaticRouting (ipv4nR2);
```

```
staticRoutenR2->AddNetworkRouteTo (Ipv4Address ("10.0.1.0"), Ipv4Mask ("/24"), Ipv4Address("10.0.3.1"), 1);
staticRoutenR2->SetDefaultRoute (Ipv4Address("10.0.3.1"), 1);

Ptr<Ipv4StaticRouting> staticRoutenTd = ipv4RoutingHelper.GetStaticRouting (ipv4nTd);

Ptr<Ipv4StaticRouting> staticRoutenR1d = ipv4RoutingHelper.GetStaticRouting (ipv4nR1d);
// Comment in to enable blackholing of nR2 on nR1d
// staticRoutenR1d->AddNetworkRouteTo (Ipv4Address ("10.0.3.0"), Ipv4Mask ("/24"), 2);

Ptr<Ipv4StaticRouting> staticRoutenR2d = ipv4RoutingHelper.GetStaticRouting (ipv4nR2d);
// Comment in to enable blackholing of nR1 on nR2d
// staticRoutenR2d->AddNetworkRouteTo (Ipv4Address ("10.0.2.0"), Ipv4Mask ("/24"), 2);

// Set variables that is used by the "services"
float cStart = 1.0;
float cStop = nPackets * 0.1 + 10;    // Add ten seconds buffer to ensure everything is transmitted
float sStop = cStop;
float pInterval = 0.01;
uint32_t pSize = 1024;

// Add randomnes to the inter-packet interval to prevent each node
// from transmitting at the same time
RngSeedManager::SetSeed (3);  // Changes seed from default of 1 to 3
RngSeedManager::SetRun (7);   // Changes run number from default of 1 to 7
Ptr<UniformRandomVariable> randPtr = CreateObject<UniformRandomVariable> ();

// Configuration of the "test" application
UdpEchoServerHelper echoServer (9);

ApplicationContainer serverApps1 = echoServer.Install (nR1);
serverApps1.Start (Seconds (cStart));
serverApps1.Stop (Seconds (cStop));

ApplicationContainer serverApps2 = echoServer.Install (nR2);
serverApps2.Start (Seconds (cStart));
serverApps2.Stop (Seconds (cStop));
```

```

    // Packets from nT to nR1 and nR2
    UdpEchoClientHelper echoClient1 (interfaceR1.GetAddress(1), 9);
    echoClient1.SetAttribute ("MaxPackets", UintegerValue (nPackets));
    echoClient1.SetAttribute ("Interval", TimeValue (Seconds (
        pInterval)));
    echoClient1.SetAttribute ("PacketSize", UintegerValue (pSize));

    UdpEchoClientHelper echoClient2 (interfaceR2.GetAddress(1), 9);
    echoClient2.SetAttribute ("MaxPackets", UintegerValue (nPackets));
    echoClient2.SetAttribute ("Interval", TimeValue (Seconds (
        pInterval)));
    echoClient2.SetAttribute ("PacketSize", UintegerValue (pSize));

    ApplicationContainer clientApps1 = echoClient1.Install (nT);
    clientApps1.Start (Seconds (cStart));
    clientApps1.Stop (Seconds (cStop));

    ApplicationContainer clientApps2 = echoClient2.Install (nT);
    clientApps2.Start (Seconds (cStart));
    clientApps2.Stop (Seconds (cStop));

    // Packets from nR1 to nR2
    UdpEchoClientHelper echoClient3 (interfaceR2.GetAddress(1), 9);
    echoClient3.SetAttribute ("MaxPackets", UintegerValue (nPackets));
    echoClient3.SetAttribute ("Interval", TimeValue (Seconds (
        pInterval)));
    echoClient3.SetAttribute ("PacketSize", UintegerValue (pSize));

    ApplicationContainer clientApps3 = echoClient3.Install (nR1);
    clientApps3.Start (Seconds (cStart));
    clientApps3.Stop (Seconds (cStop));

    // Traffic form nR2 to nR1
    UdpEchoClientHelper echoClient4 (interfaceR1.GetAddress(1), 9);
    echoClient4.SetAttribute ("MaxPackets", UintegerValue (nPackets));
    echoClient4.SetAttribute ("Interval", TimeValue (Seconds (
        pInterval)));
    echoClient4.SetAttribute ("PacketSize", UintegerValue (pSize));

    ApplicationContainer clientApps4 = echoClient4.Install (nR2);
    clientApps4.Start (Seconds (cStart));
    clientApps4.Stop (Seconds (cStop));

    ApplicationContainer extraTrafic [nClients * 4]; // Adding an
        aditional client at all nodes
```

```
  // Additional traffic between the nodes
  for(uint32_t round = 0; round < nClients; round += 1){
    UdpEchoClientHelper traficClient1 (interfaceR1.GetAddress(1), 9)
        ;
    traficClient1.SetAttribute ("MaxPackets", UintegerValue (
        nPackets));
    traficClient1.SetAttribute ("Interval", TimeValue (Seconds (
        pInterval)));
    traficClient1.SetAttribute ("PacketSize", UintegerValue (pSize))
        ;

    extraTrafic[round] = traficClient1.Install (nT);
    extraTrafic[round].Start (Seconds (cStart));
    extraTrafic[round].Stop (Seconds (cStop));

    UdpEchoClientHelper traficClient2 (interfaceR2.GetAddress(1), 9)
        ;
    traficClient2.SetAttribute ("MaxPackets", UintegerValue (
        nPackets));
    traficClient2.SetAttribute ("Interval", TimeValue (Seconds (
        pInterval)));
    traficClient2.SetAttribute ("PacketSize", UintegerValue (pSize))
        ;

    extraTrafic[round + 1] = traficClient2.Install (nT);
    extraTrafic[round + 1].Start (Seconds (cStart));
    extraTrafic[round + 1].Stop (Seconds (cStop));

    UdpEchoClientHelper traficClient3 (interfaceR1.GetAddress(1), 9)
        ;
    traficClient3.SetAttribute ("MaxPackets", UintegerValue (
        nPackets));
    traficClient3.SetAttribute ("Interval", TimeValue (Seconds (
        pInterval)));
    traficClient3.SetAttribute ("PacketSize", UintegerValue (pSize))
        ;

    extraTrafic[round + 2] = traficClient3.Install (nR2);
    extraTrafic[round + 2].Start (Seconds (cStart));
    extraTrafic[round + 2].Stop (Seconds (cStop));

    UdpEchoClientHelper traficClient4 (interfaceR2.GetAddress(1), 9)
        ;
    traficClient4.SetAttribute ("MaxPackets", UintegerValue (
        nPackets));
```

```
    traficClient4.SetAttribute ("Interval", TimeValue (Seconds (pInterval)));
    traficClient4.SetAttribute ("PacketSize", UintegerValue (pSize));

    extraTrafic[round + 3] = traficClient2.Install (nR1);
    extraTrafic[round + 3].Start (Seconds (cStart));
    extraTrafic[round + 3].Stop (Seconds (cStop));
  }



  // Print all the routing tables for debugging
  Ipv4GlobalRoutingHelper printRouting;
  Ptr<OutputStreamWrapper> routingStream = Create<OutputStreamWrapper> ("test-network.routes", std::ios::out);
  printRouting.PrintRoutingTableAllAt (Seconds (2), routingStream);

  // Packet dump of receive network
  csma.EnablePcap ("test-network", reciveNet1.Get (1), true);
  csma.EnablePcap ("test-network", reciveNet2.Get (1), true);
  // Flow monitor
  Ptr<FlowMonitor> flowMonitor;
  FlowMonitorHelper flowHelper;
  flowMonitor = flowHelper.InstallAll();

  Simulator::Stop (Seconds (sStop));
  Simulator::Run ();
  Simulator::Destroy ();

  flowMonitor->SerializeToXmlFile("test-network.xml", true, true);
  return 0;
}
```

Listing 15: test-network.cc

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
```

```
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */


/* Overveiw of the network TODO update
 *         nT
 *          |
 *nR1-nR1d-nTd-nR2d-nR2
 *
 */

#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/csma-module.h"
#include "ns3/arp-cache.h"
#include "ns3/ipv4-static-routing-helper.h"
#include "ns3/netanim-module.h"
#include "ns3/flow-monitor-helper.h"

// Normal C++ stuff
#include <iostream>
#include <fstream>
#include <string>

using namespace ns3;
using std::cout;

NS_LOG_COMPONENT_DEFINE ("NetworkDiodeTest");

int
main (int argc, char *argv[])
{
  uint32_t nPackets = 100;
  uint32_t nClients = 10;

  CommandLine cmd;
  cmd.AddValue("nPackets", "Number of packets to echo", nPackets);
  cmd.AddValue("nClients", "Number of extra clients to run", nClients);
```

```
cmd.Parse (argc, argv);

Time::SetResolution (Time::NS);
LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);

// Create the nodes
Ptr<Node> nT = CreateObject<Node> (); // node tramsittor
Ptr<Node> nR1 = CreateObject<Node> (); // first node receiver
Ptr<Node> nR2 = CreateObject<Node> (); // second node receiver
Ptr<Node> nTd = CreateObject<Node> (); // "transmitting" diode
Ptr<Node> nR1d = CreateObject<Node> (); // first "receiving" diode
Ptr<Node> nR2d = CreateObject<Node> (); // second "receiving"
   diode

Names::Add("nT", nT);
Names::Add("nR1", nR1);
Names::Add("nR2", nR2);
Names::Add("nTd", nTd);
Names::Add("nR1d", nR1d);
Names::Add("nR2d", nR2d);

NodeContainer nodesT = NodeContainer (nTd, nT);
NodeContainer nodesR1 = NodeContainer (nR1d, nR1);
NodeContainer nodesR2 = NodeContainer (nR2d, nR2);
NodeContainer nTdnR1d = NodeContainer (nTd, nR1d);
NodeContainer nTdnR2d = NodeContainer (nTd, nR2d);

// Create the network link
CsmaHelper csma;

// Create the "netwrok interfaces" and add them to the appropriate
    nodes
NetDeviceContainer sendNet;
sendNet = csma.Install(nodesT);
NetDeviceContainer reciveNet1;
reciveNet1 = csma.Install(nodesR1);
NetDeviceContainer reciveNet2;
reciveNet2 = csma.Install(nodesR2);
NetDeviceContainer netnTdnR1d;
netnTdnR1d = csma.Install(nTdnR1d);
NetDeviceContainer netnTdnR2d;
netnTdnR2d = csma.Install(nTdnR2d);

InternetStackHelper stack;
stack.Install(nodesT);
```

```
stack.Install(nodesR1);
stack.Install(nodesR2);

// Give the different networks IP-adresses
Ipv4AddressHelper address;
// Transmitt network
address.SetBase ("10.0.1.0" , "255.255.255.0");
Ipv4InterfaceContainer interfaceT;
interfaceT = address.Assign (sendNet);

// Receive network
address.SetBase ("10.0.2.0" , "255.255.255.0");
Ipv4InterfaceContainer interfaceR1;
interfaceR1 = address.Assign (reciveNet1);

// Second Receive network
address.SetBase ("10.0.3.0" , "255.255.255.0");
Ipv4InterfaceContainer interfaceR2;
interfaceR2 = address.Assign (reciveNet2);

// Network to transmit from nT to nR1
address.SetBase ("192.168.0.0", "255.255.255.252");
Ipv4InterfaceContainer intnTdnR1d;
intnTdnR1d = address.Assign (netnTdnR1d);

// Network to transmit from nT to nR2
address.SetBase ("192.168.0.4", "255.255.255.252");
Ipv4InterfaceContainer intnTdnR2d;
intnTdnR2d = address.Assign (netnTdnR2d);

// Autopopulate and generate routing tables
Ipv4GlobalRoutingHelper::PopulateRoutingTables();

// Set variables that is used by the "services"
float cStart = 1.0;
float cStop = nPackets * 0.1 + 10;    // Add ten seconds buffer to
    ensure everything is transmitted
float sStop = cStop;
float pInterval = 0.1;
uint32_t pSize = 1024;

// Add randomnes to the interval so not everything is sent at the
   same time
RngSeedManager::SetSeed (3);  // Changes seed from default of 1 to
    3
RngSeedManager::SetRun (7);   // Changes run number from default
```

```
    of 1 to 7
Ptr<UniformRandomVariable> randPtr = CreateObject<
    UniformRandomVariable> ();

// Configuration of the "test" application
UdpEchoServerHelper echoServer (9);

ApplicationContainer serverApps1 = echoServer.Install (nR1);
serverApps1.Start (Seconds (cStart));
serverApps1.Stop (Seconds (cStop));

ApplicationContainer serverApps2 = echoServer.Install (nR2);
serverApps2.Start (Seconds (cStart));
serverApps2.Stop (Seconds (cStop));

// Packets from nT to nR1 and nR2
UdpEchoClientHelper echoClient1 (interfaceR1.GetAddress(1), 9);
echoClient1.SetAttribute ("MaxPackets", UintegerValue (nPackets));
echoClient1.SetAttribute ("Interval", TimeValue (Seconds (
    pInterval)));
echoClient1.SetAttribute ("PacketSize", UintegerValue (pSize));

UdpEchoClientHelper echoClient2 (interfaceR2.GetAddress(1), 9);
echoClient2.SetAttribute ("MaxPackets", UintegerValue (nPackets));
echoClient2.SetAttribute ("Interval", TimeValue (Seconds (
    pInterval)));
echoClient2.SetAttribute ("PacketSize", UintegerValue (pSize));

ApplicationContainer clientApps1 = echoClient1.Install (nT);
clientApps1.Start (Seconds (cStart));
clientApps1.Stop (Seconds (cStop));

ApplicationContainer clientApps2 = echoClient2.Install (nT);
clientApps2.Start (Seconds (cStart));
clientApps2.Stop (Seconds (cStop));

// Packets from nR1 to nR2
UdpEchoClientHelper echoClient3 (interfaceR2.GetAddress(1), 9);
echoClient3.SetAttribute ("MaxPackets", UintegerValue (nPackets));
echoClient3.SetAttribute ("Interval", TimeValue (Seconds (
    pInterval)));
echoClient3.SetAttribute ("PacketSize", UintegerValue (pSize));

ApplicationContainer clientApps3 = echoClient3.Install (nR1);
clientApps3.Start (Seconds (cStart));
clientApps3.Stop (Seconds (cStop));
```

```
  // Traffic form nR2 to nR1
  UdpEchoClientHelper echoClient4 (interfaceR1.GetAddress(1), 9);
  echoClient4.SetAttribute ("MaxPackets", UintegerValue (nPackets));
  echoClient4.SetAttribute ("Interval", TimeValue (Seconds (
      pInterval)));
  echoClient4.SetAttribute ("PacketSize", UintegerValue (pSize));

  ApplicationContainer clientApps4 = echoClient4.Install (nR2);
  clientApps4.Start (Seconds (cStart));
  clientApps4.Stop (Seconds (cStop));

  ApplicationContainer extraTrafic [nClients * 4]; // Adding an
      aditional client at all nodes

  // Additional Packets from nT to nR1 and nR2
  for(uint32_t round = 0; round < nClients; round += 1){
    UdpEchoClientHelper traficClient1 (interfaceR1.GetAddress(1), 9)
        ;
    traficClient1.SetAttribute ("MaxPackets", UintegerValue (
        nPackets));
    //echoClient1.SetAttribute ("Interval", TimeValue (Seconds (
        randPtr->GetValue(0.0006, 0.001))));
    traficClient1.SetAttribute ("Interval", TimeValue (Seconds (
        pInterval)));
    traficClient1.SetAttribute ("PacketSize", UintegerValue (pSize))
        ;

    extraTrafic[round] = traficClient1.Install (nT);
    extraTrafic[round].Start (Seconds (cStart));
    extraTrafic[round].Stop (Seconds (cStop));

    UdpEchoClientHelper traficClient2 (interfaceR2.GetAddress(1), 9)
        ;
    traficClient2.SetAttribute ("MaxPackets", UintegerValue (
        nPackets));
    //echoClient2.SetAttribute ("Interval", TimeValue (Seconds (
        randPtr->GetValue(0.0006, 0.001))));
    traficClient2.SetAttribute ("Interval", TimeValue (Seconds (
        pInterval)));
    traficClient2.SetAttribute ("PacketSize", UintegerValue (pSize))
        ;

    extraTrafic[round + 1] = traficClient2.Install (nT);
    extraTrafic[round + 1].Start (Seconds (cStart));
    extraTrafic[round + 1].Stop (Seconds (cStop));
```

```

    UdpEchoClientHelper traficClient3 (interfaceR1.GetAddress(1), 9)
        ;
    traficClient3.SetAttribute ("MaxPackets", UintegerValue (
        nPackets));
    traficClient3.SetAttribute ("Interval", TimeValue (Seconds (
        pInterval)));
    traficClient3.SetAttribute ("PacketSize", UintegerValue (pSize))
        ;

    extraTrafic[round + 2] = traficClient3.Install (nR2);
    extraTrafic[round + 2].Start (Seconds (cStart));
    extraTrafic[round + 2].Stop (Seconds (cStop));

    UdpEchoClientHelper traficClient4 (interfaceR2.GetAddress(1), 9)
        ;
    traficClient4.SetAttribute ("MaxPackets", UintegerValue (
        nPackets));
    traficClient4.SetAttribute ("Interval", TimeValue (Seconds (
        pInterval)));
    traficClient4.SetAttribute ("PacketSize", UintegerValue (pSize))
        ;

    extraTrafic[round + 3] = traficClient2.Install (nR1);
    extraTrafic[round + 3].Start (Seconds (cStart));
    extraTrafic[round + 3].Stop (Seconds (cStop));
  }

  // Print all the routing tables for debugging
  Ipv4GlobalRoutingHelper printRouting;
  Ptr<OutputStreamWrapper> routingStream = Create<
      OutputStreamWrapper> ("test-network-no-diodes.routes", std::ios
      ::out);
  printRouting.PrintRoutingTableAllAt (Seconds (2), routingStream);

  // Packet dump of send network
  csma.EnablePcap ("test-network-no-diodes", sendNet.Get (1), true);
  csma.EnablePcap ("test-network-no-diodes", sendNet.Get (0), true);
  // Packet dump of receive network
  csma.EnablePcap ("test-network-no-diodes", reciveNet1.Get (0),
      true);
  csma.EnablePcap ("test-network-no-diodes", reciveNet1.Get (1),
      true);
  csma.EnablePcap ("test-network-no-diodes", reciveNet2.Get (0),
      true);
  csma.EnablePcap ("test-network-no-diodes", reciveNet2.Get (1),
```

```
    true);
  csma.EnablePcap ("test-network-no-diodes", netnTdnR1d.Get (0),
    true);
  csma.EnablePcap ("test-network-no-diodes", netnTdnR1d.Get (1),
    true);
  csma.EnablePcap ("test-network-no-diodes", netnTdnR2d.Get (0),
    true);
  csma.EnablePcap ("test-network-no-diodes", netnTdnR2d.Get (1),
    true);

  // Flow monitor
  Ptr<FlowMonitor> flowMonitor;
  FlowMonitorHelper flowHelper;
  flowMonitor = flowHelper.InstallAll();

  Simulator::Stop (Seconds (sStop));
  Simulator::Run ();
  Simulator::Destroy ();

        flowMonitor->SerializeToXmlFile("test-network-no-diodes.xml"
          , true, true);
  return 0;
}
```

Listing 16: test-network-no-diodes.cc

# C Snort configuration and rules

```
include ./rules/snort-nR1.rules

config daq_dir: /usr/lib/daq
config daq_mode: read-file
config daq: pcap


###################################################
# Step #5: Configure preprocessors
# For more information, see the Snort Manual, Configuring Snort - Preprocessors
###################################################

# GTP Control Channle Preprocessor. For more information, see README.GTP
# preprocessor gtp: ports { 2123 3386 2152 }

# Inline packet normalization. For more information, see README.normalize
# Does nothing in IDS mode
preprocessor normalize_ip4
preprocessor normalize_tcp: ips ecn stream
preprocessor normalize_icmp4
preprocessor normalize_ip6
preprocessor normalize_icmp6

# Target-Based stateful inspection/stream reassembly.  For more inforation, see README.stream5
preprocessor stream5_global: track_tcp yes, \
   track_udp yes, \
   track_icmp no, \
   max_tcp 262144, \
   max_udp 131072, \
   max_active_responses 2, \
   min_response_seconds 5
preprocessor stream5_tcp: log_asymmetric_traffic no, policy windows, \
   detect_anomalies, require_3whs 180, \
   overlap_limit 10, small_segments 3 bytes 150, timeout 180, \
```

```
    ports client 21 22 23 25 42 53 79 109 110 111 113 119 135 136 137 139 143 \
        161 445 513 514 587 593 691 1433 1521 1741 2100 3306 6070 6665 6666 6667 6668 6669 \
        7000 8181 32770 32771 32772 32773 32774 32775 32776 32777 32778 32779, \
    ports both 80 81 311 383 443 465 563 591 593 636 901 989 992 993 994 995 1220 1414 1830 2301 2381 2809 3037 3128 3702 4343 4848 5250 6988 7907 7000 7001 7144 7145 7510 7802 7777 7779 \
        7801 7900 7901 7902 7903 7904 7905 7906 7908 7909 7910 7911 7912 7913 7914 7915 7916 \
        7917 7918 7919 7920 8000 8008 8014 8028 8080 8085 8088 8090 8118 8123 8180 8243 8280 8300 8800 8888 8899 9000 9060 9080 9090 9091 9443 9999 11371 34443 34444 41080 50002 55555
preprocessor stream5_udp: timeout 180

# Preprocessor for snort perfomance
preprocessor perfmonitor: console
config profile_rules
config profile_preprocs
```

Listing 17: snort-nR1.conf

```
alert udp [10.0.3.0/24] any -> any any (msg: "Connection from: 10.0.3.0/24 detected"; sid:1;)
#alert udp ![10.0.3.0/24] any -> any any (msg: "Normal traffic detected"; sid:2;)
```

Listing 18: snort-nR1.rules

```
include ./rules/snort-nR2.rules

config daq_dir: /usr/lib/daq
config daq_mode: read-file
config daq: pcap

###################################################
# Step #5: Configure preprocessors
# For more information, see the Snort Manual, Configuring Snort - Preprocessors
###################################################

# GTP Control Channle Preprocessor. For more information, see README.GTP
# preprocessor gtp: ports { 2123 3386 2152 }

```

```
# Inline packet normalization. For more information, see README.normalize
# Does nothing in IDS mode
preprocessor normalize_ip4
preprocessor normalize_tcp: ips ecn stream
preprocessor normalize_icmp4
preprocessor normalize_ip6
preprocessor normalize_icmp6

# Target-Based stateful inspection/stream reassembly.  For more inforation, see README.stream5
preprocessor stream5_global: track_tcp yes, \
   track_udp yes, \
   track_icmp no, \
   max_tcp 262144, \
   max_udp 131072, \
   max_active_responses 2, \
   min_response_seconds 5
preprocessor stream5_tcp: log_asymmetric_traffic no, policy windows, \
   detect_anomalies, require_3whs 180, \
   overlap_limit 10, small_segments 3 bytes 150, timeout 180, \
    ports client 21 22 23 25 42 53 79 109 110 111 113 119 135 136 137 139 143 \
        161 445 513 514 587 593 691 1433 1521 1741 2100 3306 6070 6665 6666 6667 6668 6669 \
        7000 8181 32770 32771 32772 32773 32774 32775 32776 32777 32778 32779, \
    ports both 80 81 311 383 443 465 563 591 593 636 901 989 992 993 994 995 1220 1414 1830 2301 2381 2809 3037 3128 3702 4343 4848 5250 6988 7907 7000 7001 7144 7145 7510 7802 7777 7779 \
        7801 7900 7901 7902 7903 7904 7905 7906 7908 7909 7910 7911 7912 7913 7914 7915 7916 \
        7917 7918 7919 7920 8000 8008 8014 8028 8080 8085 8088 8090 8118 8123 8180 8243 8280 8300 8800 8888 8899 9000 9060 9080 9090 9091 9443 9999 11371 34443 34444 41080 50002 55555
preprocessor stream5_udp: timeout 180

# Preprocessor for snort perfomance
preprocessor perfmonitor
```

Listing 19: snort-nR2.conf

```
alert udp [10.0.2.0/24] any -> any any (msg: "Connection from: 10.0.2.0/24 detected"; sid:1;)
#alert udp ![10.0.2.0/24] any -> any any (msg: "Normal traffic
```

```
detected"; sid:2;)
```

Listing 20: snort-nR2.rules

# D Alternative network designs

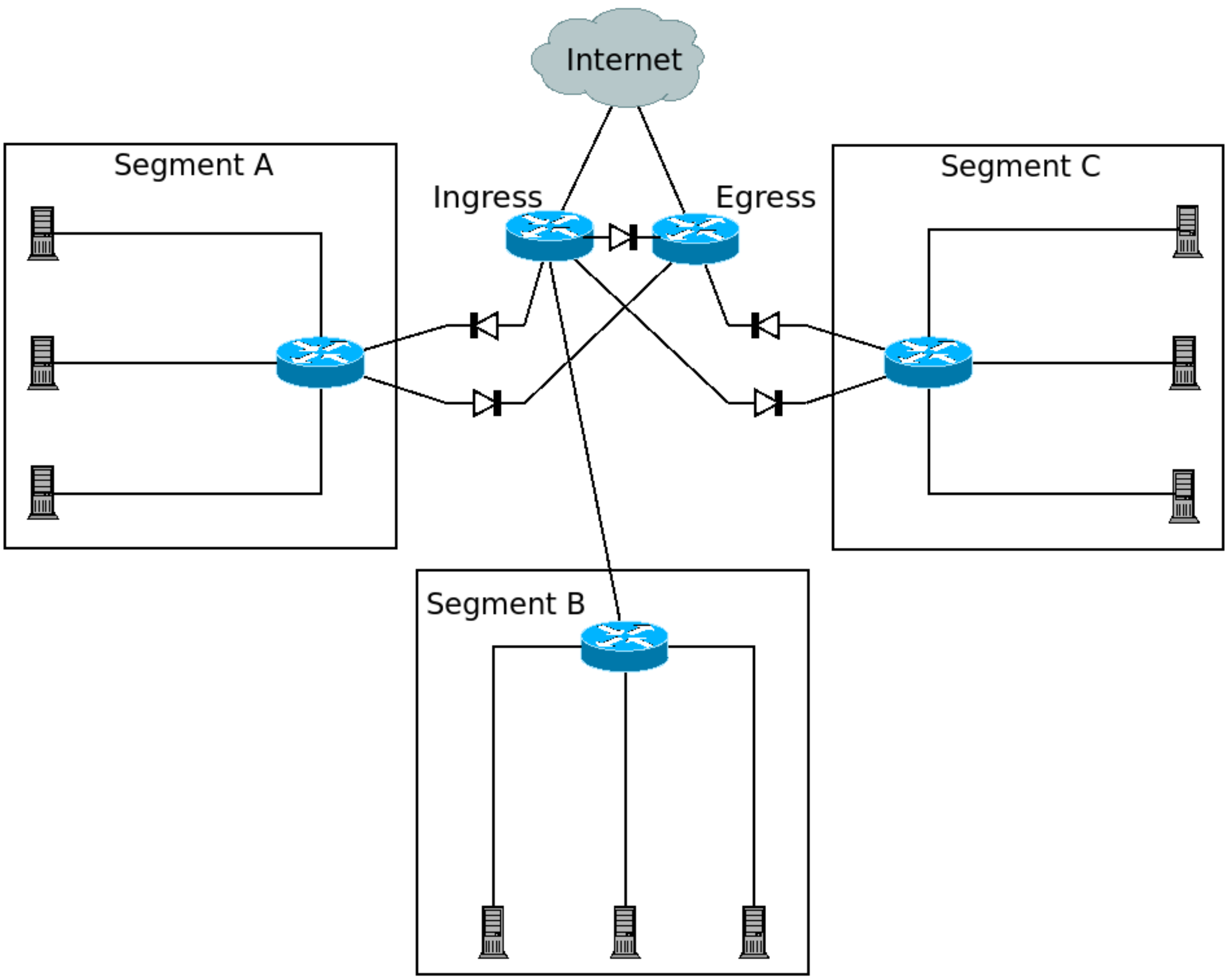



Figure 17: Alternative design of Figure 12

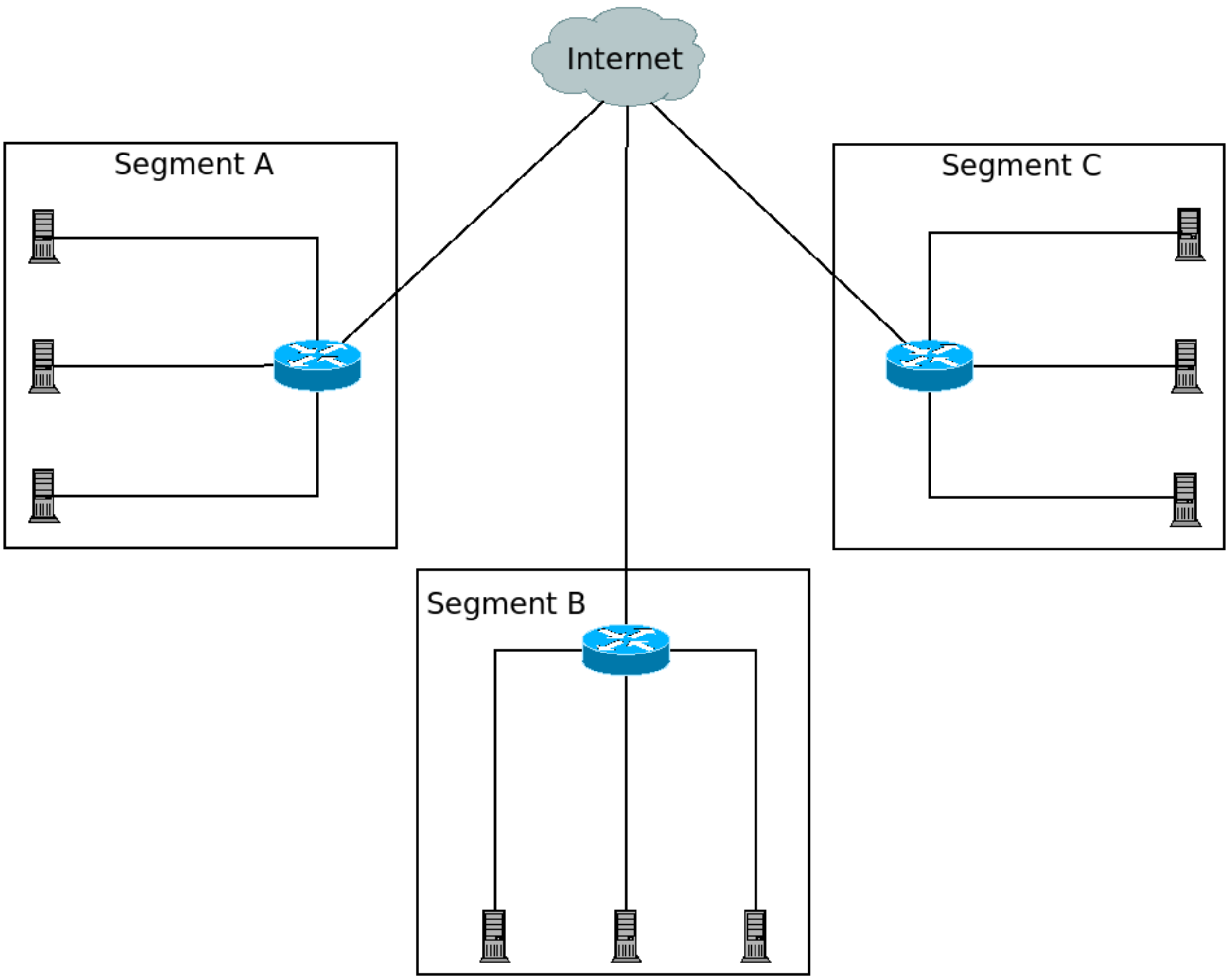



Figure 18: Alternative design of Figure 14

NTNU
Norwegian University of
Science and Technology