

BachelorTHESIS 2

Integration of a Redundancy Protocol in a SDN-based Network

Realized at Bachelor degree course

Information Technology and System-Management

University of Applied Sciences

Submitted by

Christopher Wieland

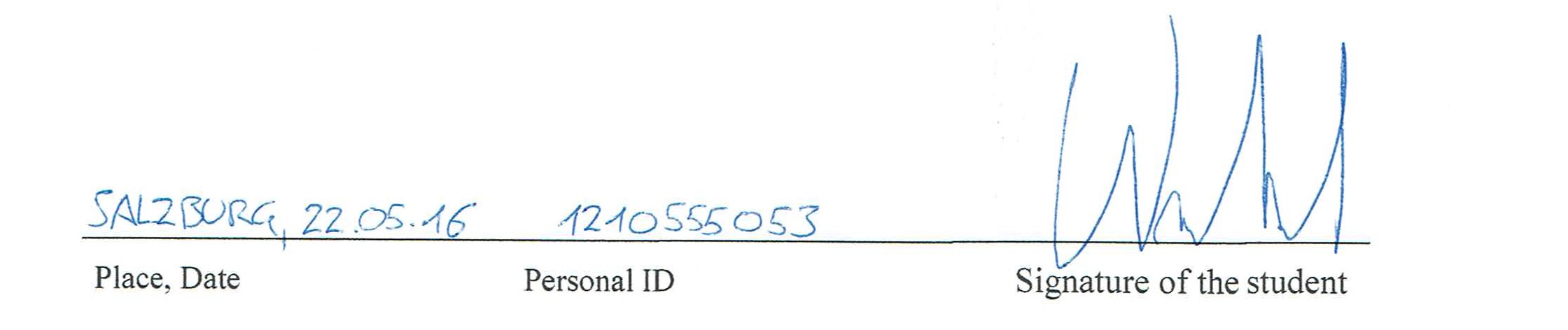
Course Director: FH-Prof. DI Dr. Gerhard Jöchtl

Advisor: DI (FH) Thomas Pfeiffenberger

Puch/Salzburg, 22.05.2016

Statutory Declaration

I declare under penalty of perjury, that I made the present bachelor thesis unassisted and without the use of other than the specified sources and all from unpublished sources, printed literature or acquired from the Internet in the text or in the substance of formulations and concepts in accordance with the guidelines scientific works cited, or have indicated precise reference to source. This work is presented in the same or similar form either at home or abroad in some form as an examination paper and is consistent with the assessed by the assessors working in agreement.



Place, Date Personal ID Signature of the student

Abstract

In existing communication networks, the overview of the whole network and the network management can be difficult. One kind of network management can be realized by using a Software-defined network (SDN). The goal in this thesis is to get reliability within a SDN-based network. To achieve the reliability, the method from a redundant protocol called High Availability Seamless Redundant Protocol (HSR) was implemented in another way than using HSR devices. The HSR protocol was chosen, because it uses media-redundancy, which can be realized. To realize the reliability by using this method, it was necessary to create a software application called duplicate packet detection. In addition, OpenFlow rules have to be set. To create the needed application, a test installation was created. The duplicate packet detection was validated for correctness. After validation, measurement is done by capturing all sent and received data in the test installation. With this information, the delay of the packets, a packet loss and the data rate were determined. The results show, that it is possible to create a reliability into a SDN-based network with OpenFlow rules and an additional application.

Summary

[1 Introduction 1](#_Toc451680932)

[2 Theoretical background 2](#_Toc451680933)

[2.1 High availability Seamless Redundancy Protocol 3](#_Toc451680934)

[2.2 Software Defined Networking 7](#_Toc451680935)

[2.3 OpenFlow 8](#_Toc451680936)

[2.3.1 Function of OpenFlow 9](#_Toc451680937)

[2.3.2 Dedicated OpenFlow Switch 10](#_Toc451680938)

[2.3.3 OpenFlow-enabled Switch 11](#_Toc451680939)

[3 Implementation of reliability in a SDN based Network 12](#_Toc451680940)

[3.1 Test arrangement of the network 12](#_Toc451680941)

[3.1.1 Used configuration for the test arrangement 13](#_Toc451680942)

[3.1.2 Used Hardware 13](#_Toc451680943)

[3.2 The strategy to create the application 14](#_Toc451680944)

[3.3 Set the OpenFlow Rules 15](#_Toc451680945)

[3.3.1 Rules for Switch #1 15](#_Toc451680946)

[3.3.2 Rules for Switch #2 16](#_Toc451680947)

[3.4 Deployment of the Java application onto the Linux based Switch 16](#_Toc451680948)

[3.4.1 Build and Run the Java application 16](#_Toc451680949)

[3.4.2 Verification of the Java application 16](#_Toc451680950)

[3.5 Deployment of the C application onto the Linux based Switch 19](#_Toc451680951)

[3.5.1 Build and Run the C application 20](#_Toc451680952)

[3.5.2 Verification of the C application 20](#_Toc451680953)

[3.6 Automated verification 23](#_Toc451680954)

[3.7 Methodology of measurements 24](#_Toc451680955)

[4 Results 25](#_Toc451680956)

[4.1.1 Measuring the Java application 25](#_Toc451680957)

[4.1.2 Measuring the C application 29](#_Toc451680958)

[5 Conclusion 33](#_Toc451680959)

[A Bibliography 34](#_Toc451680960)

[B Attachment 36](#_Toc451680961)

[B.1 Java application to detect duplicated packets 36](#_Toc451680962)

[B.2 C application to detect duplicated packets 39](#_Toc451680963)

[B.3 Script for test automatization 51](#_Toc451680964)

[B.3.1 Shell 51](#_Toc451680965)

[B.3.2 Additional Configuration File 53](#_Toc451680966)

[B.3.3 Scapy script 55](#_Toc451680967)

[B.3.4 Patheval sender and receiver script 55](#_Toc451680968)

[B.4 Matlab Script for evaluation of the data 56](#_Toc451680969)

Index of Abbreviations

SDN Software Defined Network

ACL Access Control List

QoS Quality of Service

API Application Programming Interface

HSR High Availability Seamless Redundancy Protocol

SRC Source

DST Destination

DPD Duplicate Packet Detection

API Application Programmable Interface

MAC Media Access Control

IP Internet Protocol

FCS Frame Check Sequence

ET Ethertype

SSH Secure Shell

ITU International Telecommunication Union

ETSI European Telecommunications Standards Institute

IETF Internet Engineering Task Force

ONF Open Networking Foundation

Index of Figures

[Figure 1: Example of two combined HSR Rings 4](#_Toc451505348)

[Figure 2: Ethernet Packet with implemented HSR Tag 4](#_Toc451505349)

[Figure 3: Window-sliding algorithm for duplicate packet detection 6](#_Toc451505350)

[Figure 4: Overview of a Software Defined Network Architecture with application layer 8](#_Toc451505351)

[Figure 5: An OpenFlow Switch architecture 9](#_Toc451505352)

[Figure 6: Detailed function of an OpenFlow switch 10](#_Toc451505353)

[Figure 7: Test arrangement of the network for validation and measuring 12](#_Toc451505354)

[Figure 8: Concept to create the duplicate packet detection 14](#_Toc451505355)

[Figure 9: Scenario A with Java Application 25](#_Toc451505356)

[Figure 10: Scenario B with Java Application 26](#_Toc451505357)

[Figure 11: Scenario C with Java Application 26](#_Toc451505358)

[Figure 12: Scenario D with Java Application 27](#_Toc451505359)

[Figure 13: Scenario E with Java Application 28](#_Toc451505360)

[Figure 14: Comparison of delay from all measurements with Java application 28](#_Toc451505361)

[Figure 15: Scenario A with C Application 29](#_Toc451505362)

[Figure 16: Scenario B with C Application 30](#_Toc451505363)

[Figure 17: Scenario C with C Application 30](#_Toc451505364)

[Figure 18: Scenario D with C Application 31](#_Toc451505365)

[Figure 19: Scenario E with C Application 31](#_Toc451505366)

[Figure 20: Comparison of delay from all measurements with C application 32](#_Toc451505367)

Code-Snippet-Index

[Script 1: Added code for verification in java application 17](#_Toc451505368)

[Script 2: Added code for verification in C application 21](#_Toc451505369)

[Script 3: Java Application to detect duplicate packets 39](#_Toc451505370)

[Script 4: C Application to detect duplicate packets 50](#_Toc451505371)

[Script 5: Shell Script for test automatization 53](#_Toc451505372)

[Script 6: Additional Configuration File for Shell Script 55](#_Toc451505373)

[Script 7: Scapy script to send data 55](#_Toc451505374)

[Script 8:Patheval sender script 56](#_Toc451505375)

[Script 9: Patheval receiver script 56](#_Toc451505376)

[Script 10: MATLAB script for evaluation 70](#_Toc451505377)

Index of Tables

[Table 1: Information of the test arrangement 13](#_Toc451505378)

[Table 2: Used material for the sender 13](#_Toc451505379)

[Table 3: Used material for the receiver 13](#_Toc451505380)

[Table 4: Used material for switch #1 14](#_Toc451505381)

[Table 5: Used material for switch #2 14](#_Toc451505382)

[Table 6: Results of the java application verification with Ethernet packets 17](#_Toc451505383)

[Table 7: Results of the java application verification with TCP packets 18](#_Toc451505384)

[Table 8: Results of the java application verification with UDP packets 18](#_Toc451505385)

[Table 9: Results of the java application verification with ICMP packets 19](#_Toc451505386)

[Table 10: Results of the c application verification with Ethernet packets 21](#_Toc451505387)

[Table 11: Results of the c application verification with TCP packets 22](#_Toc451505388)

[Table 12: Results of the c application verification with UDP packets 22](#_Toc451505389)

[Table 13: Results of the c application verification with ICMP packets 23](#_Toc451505390)

[Table 14: Methodology of measurements 24](#_Toc451505391)

# Introduction

There are different ways especially in the areas of performance and critical section to build a network. One way of realization is by using the Software-Defined Network (SDN) network management. In ordinary networks, reliability can be realized by using redundancy protocols like Parallel Redundancy Protocol (PRP) or High Availability Seamless Redundancy Protocol (HSR) with expensive customized hardware. The goal of this thesis is to realize a reliability in a SDN managed network by creating a method, which use OpenFlow Rules and an additional software application.

For creating the method, the strategy of the method from the HSR protocol is used to create the reliability in a SDN managed network.

To create the method, it was necessary to use OpenFlow rules and an additional application, which can detect duplicate packets.

After creating and validation of the method, measurements was done to verify the correctness of the method and to test of stability of the created method.

# Theoretical background

This part of the thesis will be elucidating the Software-Defined Network (SDN). SDN is a networking method, which allows managing the network services by abstraction control and data plane. The control plane or also called controller is able to make the network easier to manage. The data plane has the responsibility to forward the network traffic to the destinations.

The controller is the centralized part in a SDN managed network. The controller get connection by using the Southbound-API to the devices in the data plane. With the Westbound-API, there is a possibility to connect the controller with other controllers from an internal SDN based network. The controller has the opportunity to send instructions to the devices in the data plane. These instructions can be set from applications in an optional application layer. With this opportunity, it is possible to write own programs for the devices [1].

SDN also uses the concept of bounds. North-, West-, East- and Southbound APIs are important in software-defined networks. These bounds are all at ITU-T with the reference ITUSG11 standardized. The coherence of this bounds and SDN is widely present. The Northbound-API in a SDN managed network is very important. Because they are so critical, Northbound-APIs must support a wide variety of applications. Without this bound, the Controller could not command the devices with external applications in the network and management would not be possible [2].

The Eastbound-API has the responsibility to make communication between a SDN managed network and an external non-SDN managed network possible. Proper function is dependent on the legacy network. The non-SDN network has to implement a translation tool for communication. If there is a translation tool in the legacy-technology network, both networks should ideally appear to be fully compatible to each other. With Westbound‐API an information channel between several internal networks are given. This API provides the information exchange of network state, routing decisions and allows the configuration of network flow via several networks [3].

Southbound APIs facilitate efficient control over the network and enable the SDN controller to make changes dynamically according to real-time demands and needs.

The Southbound‐API represents the interface between control‐ and data plane. In view of that, this API is essential for the SDN principle [3]. OpenFlow was developed by the Open Networking Foundation (ONF) and is the first and probably most well known Southbound-API [2]. OpenFlow is one of the most used Southbound –APIs and is used as such one in this thesis.

The main part of this thesis is to create reliability in a network. For recreating the used method of the High Availability Seamless Redundancy protocol (HSR), OpenFlow rules and an additional software application is needed.

This software application is named duplicate packet detection (DPD) should detect all duplicate packets in a media-redundant network without adding or changing information of the packet. The duplicate packet detection has the responsibility to sort out all duplicate packets and forward all non-duplicate packets to the receiver.

## High availability Seamless Redundancy Protocol

The High Availability Seamless Redundancy (HSR) Protocol, which is also a development of the Parallel Redundancy Protocol, is an Ethernet Redundancy Protocol. IEC 62349-3 is the reference for the standardization of the HSR Protocol. In comparison to the Parallel Redundancy Protocol, which works with network-redundancy, this protocol creates media-redundancy by sending the original packet and a copied packet on both interfaces.

On media-redundancy, the data are sent redundant on more than one link. An example would be a ring topology. In a ring- topology, the data are sent clockwise and anti-clockwise. If one link breaks down, the packet will reach the receiver from the redundant link. In comparison to media-redundancy, network-redundancy send the data through more than one infrastructure. [4].

By HSR Protocol, ring topologies are in use. In every HSR Ring, there can be four different nodes: a Double Attached Node for HSR (DANH), a Single Attached Node (SAN), RedBoxes and QuadBoxes, which can be seen in Figure 1.

DANH is able to connect with two devices in the ring, while SAN can only connect one device. By HSR, it is not possible to connect SAN directly to the network, so there is a need to use RedBoxes. The use of RedBoxes are important, because these devices are redundancy proxies in a network, which use HSR. The function of QuadBoxes are to connect two different HSR rings [4].

Figure 1 shows an example of such a HSR ring:

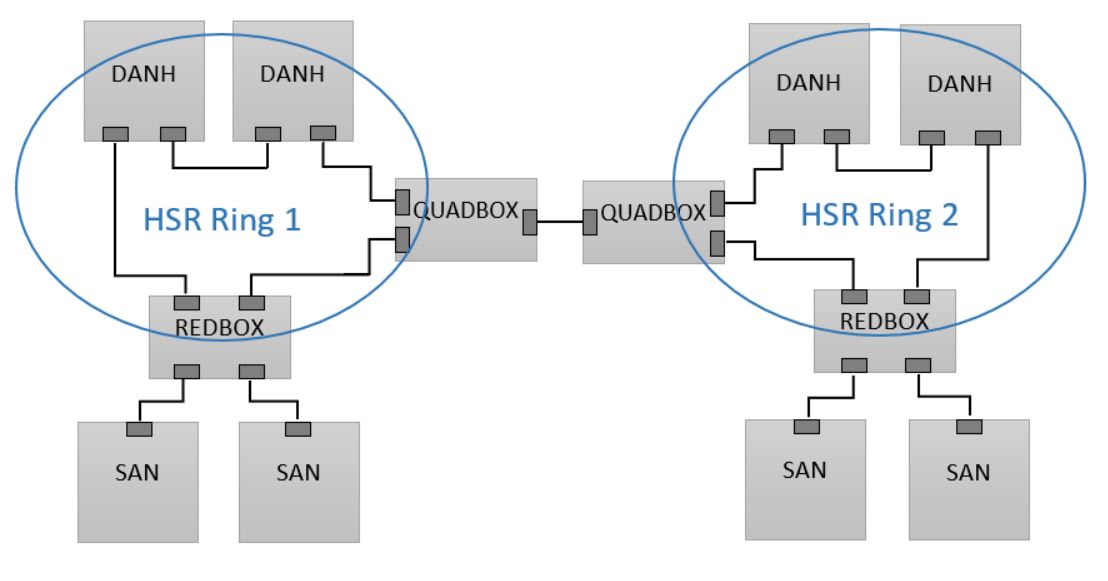


Figure 1: Example of two combined HSR Rings

It is not possible to connect devices with only one interface to the HSR ring. All devices with only one network interface are SAN. All other devices in a HSR ring have at least two network interfaces, where every interface per device has the same MAC-address and the same IP address. SANs also cannot read the packages without the RedBoxes, because the HSR method add a tag in the middle of the packet.

Therefore, the packet is only readable by the devices, which are in the ring. This HSR frame is the HSR tag. In this tag, there is the following information represented:

* the length of the payload,
* the switch port from the source device and

the sequence number of the packet. Figure *2* shows an Ethernet packet with the added HSR tag:

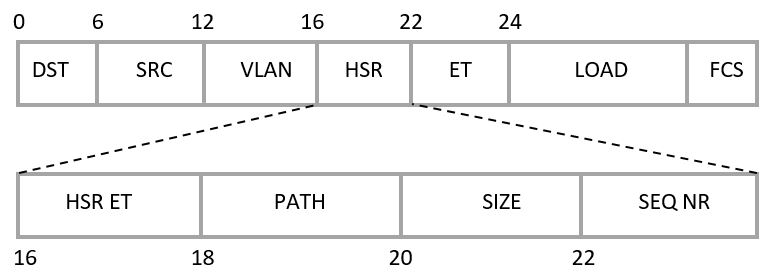


Figure 2: Ethernet Packet with implemented HSR Tag

Figure 2 shows, that 6 Bits shift the Ethertype (ET), the payload (LOAD) and the frame check sequence (FCS). Because of this shifting, the packet would be marked as corrupt in non-HSR devices. Only HSR devices like RedBox, DANH or QuadBoxes are able to read the packet [5].

As mentioned at the beginning of this chapter, HSR uses media-redundancy. The meaning about this is that each packet, which should be sent through the network, will be sent twice, one clockwise and the other packet anti-clockwise.

To prevent redundant packets on the receiver, there are two parallel prevention loops used. The first prevention loop is based on network traffic characteristics on the sender and receiver node [5].

The second prevention loop is based on the duplicate packet detection algorithms. For duplicate packet detection, there can be used two algorithms [5].

The first algorithm is called puffer algorithm. The sequence number in the HSR tag is checked at every incoming packet. If the sequence number of the incoming packet is not already in the puffer, the packet will be forwarded to the destination. After forwarding to the receiver, the sequence number of the packet will be copied into the puffer. If a sequence number of another packet matches one of the sequence numbers in the puffer, the packet identified as duplicated and will be dropped.

The biggest disadvantage of this method is the size of the puffer. The larger the puffer is the slower the transmission rate will be. The explanation to this disadvantage is that the DPD need to search in every slot in the puffer, whether the new sequence number matches one of the numbers in the Puffer, which need a lot of time. To pretend this disadvantage, an additional function can be added, which keep the puffer size small [5].

The second method is called “sliding-windows algorithm”.

“*One version of the sliding window algorithm was a mandatory part of the original PRP-0 specification in IEC 62439: Edition 1. PRP-0 and subsequently the mandatory use of the Sliding Window Algorithm was deprecated with the release of IEC 62439-3: Edition 2, where PRP-0 was replaced by PRP-1.*” [5]

This method works on the assumption, that the HSR protocol always increments the sequence number in the HSR Tag of the packets by one. In addition, it is needed to set up the number of windows for the devices and the window size of each device. The following figure shows a sketch of this algorithm:

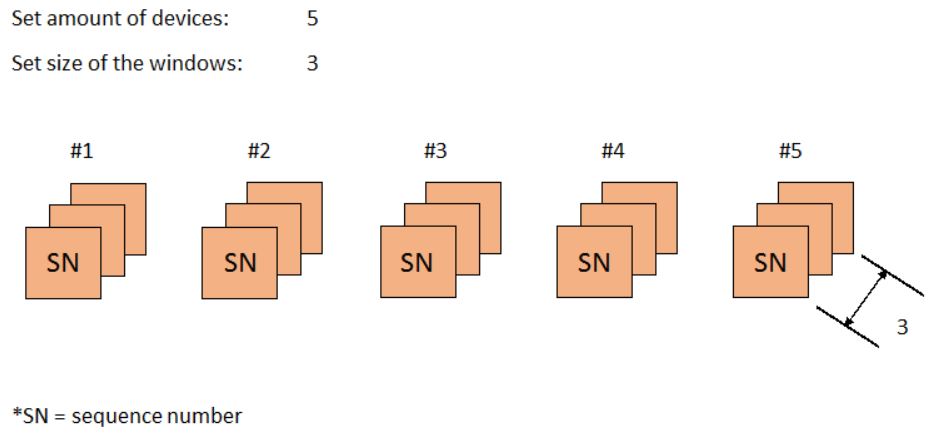


Figure 3: Window-sliding algorithm for duplicate packet detection

Each device in the network will get one window. The first step in this method is to sort the windows by the MAC-Address of the devices. If a packet will be send to the receiver, the sequence number of this packet is shown in the window of the device. If the sequence number of the new packet is the highest number, the window of the device will be queued to the first place [5].

One advantage of the algorithm is the reduced memory use, compared to the puffer method. In the window-sliding algorithm, there is a fix memory usage, while by the puffer algorithm the puffer gets larger. A disadvantage of this method is the higher implementation effort, because of the restriction to a limited number of windows. If the size of the windows is set static and there are more HSR devices than the count of windows, the duplicate packet detection will not work for these devices, because there are no windows left. Furthermore, the windows should be big enough. If the set window size of a device is smaller than the sent packets from the device, the DPD also does not work [5].

## Software Defined Networking

The beginning of creation for a concept of SDN was in 1996 and has multiple standard bodies. Institutions, which develop and extend this concept, are the Internet Engineering Task Force (IETF), the International Telecommunication Union (ITU), the Open Networking Foundation (ONF) and the European Telecommunications Standards Institute (ETSI) [6].

SDN is an approach to manage networks. It is a concept, which separates the data plane from the control plane. The data plane has the responsibility to forward the packets through the network to the destination. The control plane has the responsibility to manage the network by using APIs. In the application layer, there can be several applications, which use the controller to manage the devices in the network.

With this abstraction, the network management will be much easier, because of a better overview. The usage of SDN is based on simplification and administration of the whole network, because of the opportunity to manage the network with management tools and add additional applications, which can used by the controller [2].

The control plane, or controller, is the central part of SDN architecture. The controller is needed to send information and instructions to the data plane. This instruction can be set in applications in the application layer, which is connected via the Northbound-API to the controller [2].

Figure 4 gives an overview about the abstraction with SDN.

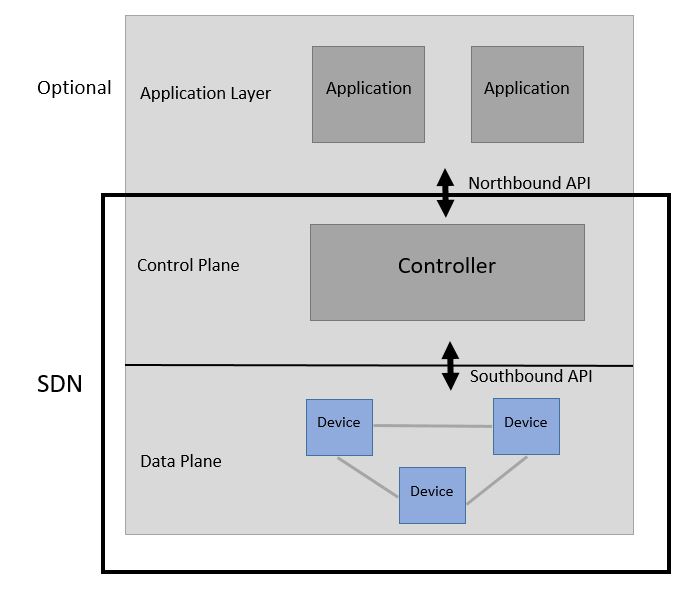


Figure 4: Overview of a Software Defined Network Architecture with application layer

Figure 4 shows the abstraction of control and data plane. It also shows how the SDN architecture works. The controller is, as shown in Figure 4, the centralized unit in the SDN architecture. The controller is able to gather information from the applications, which are implemented in the application layer and send it to the devices in the network. With this separation, the network management is simplified.

Furthermore, with the application layer the Software Defined Network has the opportunity to use external applications. These external applications could be network management applications, analytic applications or business applications. With these services of SDN, there is no bond to any manufacturer [2].

## OpenFlow

OpenFlow is one of the most common Southbound protocols. There are different implementations of an OpenFlow controller, like Open Daylight, ONOS, and many others [2]. OpenFlow is a communication protocol, which was created in March 2008 and is still being extended.

Open Networking Foundation (ONF) administers this communication protocol. OpenFlow provides an open protocol [7]. The OpenFlow protocol allows controlling the switch more than one controller for increasing the robustness and the performance.

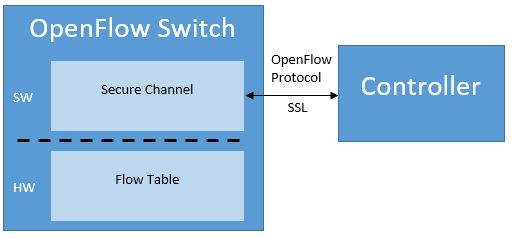
  
Figure 5, shows an OpenFlow Switch.

Figure 5: An OpenFlow Switch architecture

An OpenFlow managed switch consists of three parts: A flow-table, a secure channel and the OpenFlow protocol.

* On the flow table, there are actions associated with each flow entry, which are extensible.
* The secure channel connects the switch with a control unit, allowing commands for sending packets between the controller.
* The switch and the OpenFlow protocol provides a way for the controller to communicate with the switch.

### Function of OpenFlow

In Figure 6 there is shown, how an OpenFlow managed switch acts by receiving a new packet. If the sender sends a packet to the receiver and there is no rule set in the flow table for such a packet, the controller has to make a new instruction for this kind of packets.

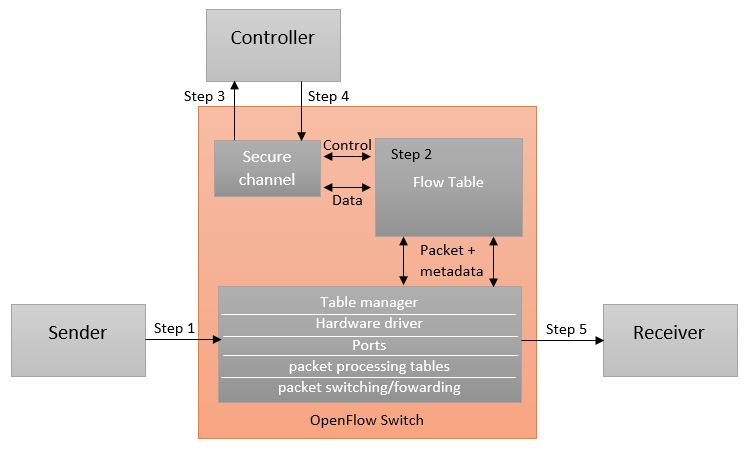


Figure 6: Detailed function of an OpenFlow switch

This figure shows the case, that there is no rule set in the flow table for the packets, which are send by the sender. In step 1, the sender sends a packet to the OpenFlow switch. In the next step, the switch checks in the flow table. If there is no rule in the table, the packet will be sent via a secure OpenFlow channel to the controller (Step 3). In step 4, the controller executes an instruction for this kind of packet and sends the packet with the instruction back to the switch. In the last step, the instruction will be set into the flow table and the packet will be sent to the receiver [8].

There are two types of OpenFlow switches: A dedicated OpenFlow switch and an OpenFlow-enabled switch [7].

### Dedicated OpenFlow Switch

A dedicated OpenFlow switch is a dumb device that only forwards all packets between the ports defined by the controller. The only limitations by using OpenFlow is by creating rules into the flow table. In a flow entry, there could be recorded different things for example:

* A UDP connection,
* packet forwarding by the same VLAN Tag or
* forwarding packets between two given ports.

Every flow entry has a simple action attached to it. There are three basic actions:

* Forward the flows to a given port or more ports. With this action, all packets are send to the destination.
* Encapsulate and forward a packet to the controller. There is a transportation for the packets through a secure SSL connection to the controller. This happens, if the first packet of a flow is received.
* Drop all packets in this flow. This is used for security or for reducing the traffic in the network.

An entry in the flow-table has three fields: A packet header, which defines the flow, the action that shows what will happen with the packets and the statistics, which shows the number of packets from the flow, the bytes and the time, the last packet has matched the entry [7].

### OpenFlow-enabled Switch

This kind of switches are commercial devices, which are enhanced with OpenFlow features by adding the flow-table, secure tunnel and the OpenFlow protocol. Switches with the basic actions as mentioned above, are called “Type 0” devices. Devices, which has additional features, like rewriting portions of the packet header, are called “Type 1” devices [7].

# Implementation of reliability in a SDN based Network

The goal in this thesis is to reach reliability in a SDN based network with OpenFlow rules and an additional software application called duplicate packet detection. The method of the High Availability Seamless Redundancy Protocol is choosing to imitate, because it is one of the newest protocols and is known as protocol, which has no packet-loss by interruption. It works with media-redundancy, which is manageable to emulate with an application and OpenFlow rules. Another point is that HSR is the best choice in consideration of safety and stability in comparison to the other protocols. The OpenFlow rules are needed to forward the packet through the network. For this case, in each switch, which is used in this thesis rules are added.

To achieve reliability, it was necessary to create an application, which can detect duplicated packets. This application need a virtual interface to run correctly. In chapter 5 there is mentioned, how it could be possible, that the duplicate packet detection is able to sniff at more than one interface at the same time. At this point, a virtual interface is not necessary anymore.

## Test arrangement of the network

Figure 7 shows the test arrangement of a network:

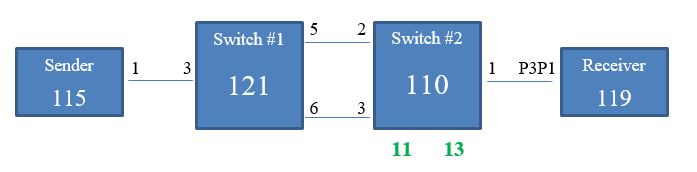


Figure 7: Test arrangement of the network for validation and measuring

With this test arrangement, it is possible to simulate a network, which sends packets on two ways. The software application in combination with OpenFlow rules, which is mentioned in chapter 2, works on switch #2. Based on OpenFlow rules the packets received on Port 3 of switch #1, will be copied and forwarded to port 5 and 6. With this construction, switch #2 gets all packets twice and the duplicate packet detection can be tested.

### Used configuration for the test arrangement

The Table 1 below shows the additional data to the Testbed:

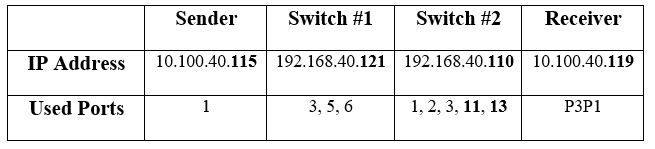


Table 1: Information of the test arrangement

To realize the concept of the thesis, it is necessary to work with a virtual port. For realization, on switch #2 one virtual port (11) was added to test the applications. For the software development, another virtual port (13) was created to capture all dropped packets.

### Used Hardware

|  |  |  |
| --- | --- | --- |
| Sender | | |
| PC | Lenovo Thinkcentre |  |
| OS | Ubuntu 14.04.3 LTS |  |

Table 2: Used material for the sender

|  |  |
| --- | --- |
| Receiver | |
| PC | Lenovo Thinkcentre |
| OS | Ubuntu 14.04.3 LTS |

Table 3: Used material for the receiver

|  |  |
| --- | --- |
| Switch #1 | |
| Switch | Cubro EX2 |
| VERSION | 1.3.2.2 |
| OVS | 1.9.90 |

Table 4: Used material for switch #1

|  |  |
| --- | --- |
| Switch #2 | |
| PC | Lenovo Thinkcentre |
| OS | Ubuntu 14.04.3 LTS |
| OVS | 2.5.0 |

Table 5: Used material for switch #2

## The strategy to create the application

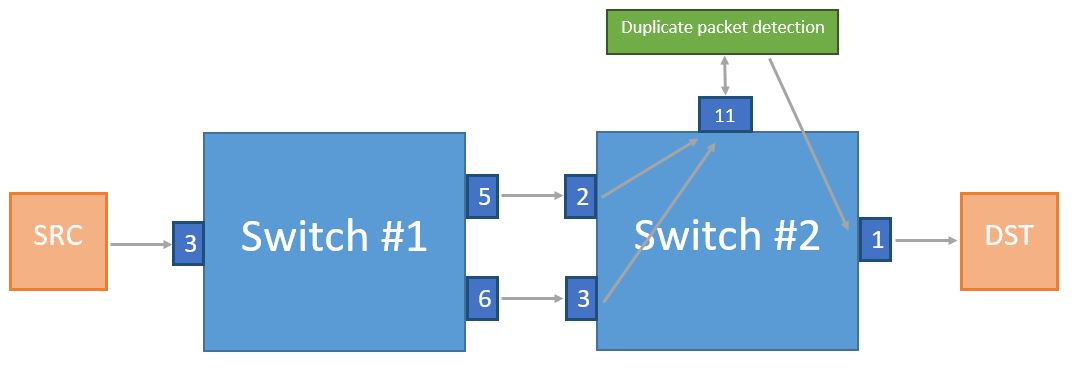
The idea is a modified strategy from the HSR duplicate packet detection. In this case, the packet itself has no additional tag for identification. The payload will identify the packets in this way.

Figure *8* shows a concept to create a duplicate packet detection application.

Figure 8: Concept to create the duplicate packet detection

For a proper function of the application, it is now necessary to create one virtual port on Switch #2. In next steps, by editing the duplicate packet detection, it could be possible to gather the packets from Port 2 and Port 3 at the same time by using of multithreading. A packet should be send from source (SRC) to destination (DST). Switch #1 has the responsibility to copy all packets, which are received on Port 3 and send them to Port 5 and Port 6. In this example, Port 2 and Port 3 from switch #2 gather the packets.

The virtual Port 11 gets the packets from Port 2 and Port 3 via OpenFlow rules. Now there is a decision in the local process: if the packet is not a duplicate one, the packet will be sent to Port 1. From Port 1 the packet will be delivered to the destination. It was necessary that there is no modification of the packet, but it has to be noticed as duplicated. To achieve this goal, the payload of every packet are captured in a puffer.   
The payload of the next packet will be compared with all payload entries in the puffer from the packets before. If the payload is identical with one of the payload in the puffer, then it is a duplicated one and it will be dropped. If it is not in the puffer, the payload of the packet will be added into the puffer and the packet will be send to Port 1.

## Set the OpenFlow Rules

To setup the Testbed, there was a need to set up OpenFlow rules on all switches.

The following commands are used to add rules to the devices. For example:

“ovs-ovsctl add-flow br0 “in\_port=6,actions=output:3” allows to send all packets, which are received on port 6 to port 3. On switch #2 it was needed to put extra some rules for the arp connection. This arp rules are needed for the communication of the devices. On chapter 3.3.2, there can be seen arp rules.

### Rules for Switch #1

The rules are the following:

ovs-ovsctl add-flow br0 “in\_port=3,actions=output:5,6”

ovs-ovsctl add-flow br0 “in\_port=6,actions=output:3”

### Rules for Switch #2

The rules on this switch are:

ovs-ovsctl add-flow br0 “in\_port=1,actions=output:11”

ovs-ovsctl add-flow br0 “in\_port=2,actions=output:11”

ovs-ovsctl add-flow br0 “in\_port=2,arp,actions=output:1”

ovs-ovsctl add-flow br0 “in\_port=1,arp,actions=output:3”

## Deployment of the Java application onto the Linux based Switch

To implement the java application, there are two steps needed to do. The first step is to install an external library called “jNetPcap” onto the switch.

The last step is to copy the whole java application onto the switch. In this case, the program is called “packetdetection” and is copied to the “/root/packetdetection/Java/” directory.

To verify the correctness and proper function of the application, the duplicate packet detection software was tested in the test arrangement and additional capture files are created with tcpdump on the sender, receiver and switch #2.

### Build and Run the Java application

To execute the application, the following commands have to be typed into the console:

“cd /root/packetdetection/Java/packetdetection/src”

“javac –cp /root/packetdetection/jnetpcap/jnetpcap.jar detection.java”

“java –cp .:/root/packetdetection/jnetpcap/jnetpcap.jar detection 1 2”

To choose the right interface, the application prints out all existing devices with the needed numbers. “1, 2” represent de devices, which should be chosen to sniff and send the packet. The first letter stands for the sniffing device and the second letter stand for the interface where the packet should be sent.

### Verification of the Java application

For verification of the duplicate packet detection, a second virtual interface was added in switch #2 called Eth 13. This interface is used to capture all dropped packets. In the java application, there was added the following code:   
  
System.out.printf("\nChoose a senddevice!\n"); //Set a device to send

int choosedevicesend = consoleinput.nextInt();

PcapIf dropdevice = alldevices.get(choosedevicedrop);

System.out.printf("\n '%s' is chosen to get packet\n",(dropdevice.getDescription() != null)?dropdevice.getDescription(): dropdevice.getName());

final Pcap drop\_if = Pcap.openLive(dropdevice.getName(), snaplen, flags, timeout, errbuf);

Script 1: Added code for verification in java application

To verify the duplicate packet detection, it was necessary to test it with different protocols. For the tests, the protocols TCP, UDP, ICMP and additional one test with raw Ethernet packets was done. For this test, a software called Scapy is used. Scapy will be explained in more detail in chapter 3.6. To test the duplicate packet detection, it was necessary to build packets with the same payload via Scapy. By each protocol, there are sent 50 packets. If the duplicate packet detection works correctly, it should only send one packet and drop the other 49, because they are duplicate packets.

In the following tables, there are the results of the verification tests for each tested Protocol.

The next table shows the results of the verification test with raw Ethernet packets.

|  |  |  |
| --- | --- | --- |
| Ethernet frames | Interfaces | counted packets |
| Switch #2 | Eth 1 | 1 |
| Eth 11 | 50 |
| Eth 13 | 49 |

Table 6: Results of the java application verification with Ethernet packets

Table 6 shows, how many packets are sent and received. On switch #2 the duplicate packet detection is running. Eth 1 is the interface, which send the packets to the receiver. Eth 11 receive all packets and Eth13 drop all duplicate packets.

The java application has forwarded the first packet, and dropped all other, because of the same payload. With this short test, it is verified that the duplicate packet detection works correctly with raw Ethernet packets.

The next Table shows the results of the verification with TCP packets.

|  |  |  |
| --- | --- | --- |
| TCP | Interfaces | counted packets |
| Switch #2 | Eth 1 | 1 |
| Eth 11 | 50 |
| Eth 13 | 49 |

Table 7: Results of the java application verification with TCP packets

As discussed above with Table 6, Table 7 also shows correct results with TCP packets. One packet was sent to the receiver and the other duplicate packets are dropped.

Table 8 shows the results of the verification with UDP packets.

|  |  |  |
| --- | --- | --- |
| UDP | Interfaces | counted packets |
| Switch #2 | Eth 1 | 1 |
| Eth 11 | 50 |
| Eth 13 | 49 |

Table 8: Results of the java application verification with UDP packets

The results of the verification with UDP packets correct. The application detects the first packet and send it to the receiver. The second and each subsequent packet will be detected as duplicate packet and will be dropped.

The last verification test is with ICMP packets.

|  |  |  |
| --- | --- | --- |
| ICMP | Interfaces | counted packets |
| Switch #2 | Eth 1 | 1 |
| Eth 11 | 50 |
| Eth 13 | 49 |

Table 9: Results of the java application verification with ICMP packets

As expected, also the test with ICMP packets shows a correct function.

The verification was done with the most common protocols and the results show a correct function. With this verification, the application can be safely used for the measurement.

## Deployment of the C application onto the Linux based Switch

To implement the c version of the duplicate packet detection, it is necessary to install an external library called “lpcap” onto the switch. To install lpcap, the latest version should be downloaded by typing the following command:

“apt-get install libpcap0.8 libpcap0.8-dev libpcap-dev”

The last step is to copy the whole application onto the switch. In this case, the program is called “packetdetection” and is copied to the “/root/packetdetection/C/” directory.

For the creation of the c application, it was necessary to do some changes, which differ from the basic idea about send and drop a packet. In comparison to jNetPcap, which is used in the java application as external library, the library libpcap in the c application does not support sending captured packets. To create a workaround, it was necessary to create a raw socket. With this raw socket, it is possible to send the captured packets to the interface.

To verify the correctness and proper function of the application, the application was tested in the test arrangement and capturing files was created with tcpdump on the sender, receiver and switch #2.

### Build and Run the C application

To execute the application, the following commands have to be typed into the console:

“cd /root/packetdetection/C/packetdetection/src”

“gcc -c dpd.c -static –lpcap”

“gcc -o dpd dpd.o –lpcap”

“./dpd 100 0 eth11 eth1 “0xD4 “0x3D “0x7E” “0xA8” “0xE4” “0xCC” ”

The first parameter of the execution command is to set the size of the puffer, which is used in the program to gather the payload data of each packet, in this case, it is set to 100. The second parameter is to set a logging level. If the logging integer is set to “1”, all payload data will be extra logged into a .txt file, if the integer is “0”, then there will be no logging active. Eth11 and Eth1 represents the devices, which should be chosen to sniff and send the packet. Eth11 represents the sniffing device and Eth1 represents the interface where the packet should be sent. The hex values, which are also appended on the execution command are needed to set the mac address of the sending device. In this case, the sending device Eth1 has the following MAC Address: D4:3D:7E:A8:E4:CC. These addresses are used to transport the new created packet to the right device.

### Verification of the C application

To verify the c version of the duplicate packet detection, the second virtual interface from switch #2, called Eth 13, was used again. The following code was added:

const char\* DROPDEVICE;

char\* DROPDEVMAC0, DROPDEVMAC1, DROPDEVMAC2, DROPDEVMAC3, DROPDEVMAC4, DROPDEVMAC5;

int dropping;

dropping = socket(AF\_PACKET, SOCK\_RAW, IPPROTO\_RAW);

memset(&if\_idx, 0, sizeof(struct ifreq));

strncpy(if\_idx.ifr\_name, DROPDEVICE, IFNAMSIZ-1);

if (ioctl(dropping, SIOCGIFINDEX, &if\_idx) < 0) perror("SIOCGIFINDEX");

socket\_address.sll\_ifindex = if\_idx.ifr\_ifindex;

socket\_address.sll\_halen = ETH\_ALEN;

socket\_address.sll\_addr[0] = (long)DROPDEVMAC0;

socket\_address.sll\_addr[1] = (long)DROPDEVMAC1;

socket\_address.sll\_addr[2] = (long)DROPDEVMAC2;

socket\_address.sll\_addr[3] = (long)DROPDEVMAC3;

socket\_address.sll\_addr[4] = (long)DROPDEVMAC4;

socket\_address.sll\_addr[5] = (long)DROPDEVMAC5;

if((senddata = malloc(strlen(addrdata)+strlen(dataload)+1)) != NULL){

memset(senddata, 0, strlen(addrdata)+strlen(dataload)+1);

strcat(senddata,addrdata);

strcat(senddata,dataload);

}

if (sendto(dropping, senddata, strlen(addrdata)+strlen(dataload)+1, 0, (struct sockaddr\*)&socket\_address, sizeof(struct sockaddr\_ll)) < 0) printf("Drop failed\n");

free(senddata);

}

Script 2: Added code for verification in C application

To verify the c application, the same tests as for the java application in chapter 3.4.2 were executed.

As in chapter 3.4.2, the following tables show the results of the verification tests for each tested Protocol.

|  |  |  |
| --- | --- | --- |
| Ethernet frames | Interfaces | counted packets |
| Switch #2 | Eth 1 | 1 |
| Eth 11 | 50 |
| Eth 13 | 49 |

Table 10: Results of the c application verification with Ethernet packets

Table 10 shows, how many packets are sent and received. The same test for verification was made in chapter 3.4.2**Fehler! Verweisquelle konnte nicht gefunden werden.** with the java application. The C application has sent the first packet, and dropped all other. With the short test it is verified again, that this application works correctly with raw Ethernet packets.

In the next table, the results of the verification with TCP packets are shown.

|  |  |  |
| --- | --- | --- |
| TCP | Interfaces | counted packets |
| Switch #2 | Eth 1 | 1 |
| Eth 11 | 50 |
| Eth 13 | 49 |

Table 11: Results of the c application verification with TCP packets

As expected, the c application also separates the normal packets from the duplicate packets by dropping all duplicates.

Table 12 shows the results of the verification with UDP packets by using the c application.

|  |  |  |
| --- | --- | --- |
| UDP | Interfaces | counted packets |
| Switch #2 | Eth 1 | 1 |
| Eth 11 | 50 |
| Eth 13 | 49 |

Table 12: Results of the c application verification with UDP packets

The packets, which was sent with the UDP protocol, was sorted out correctly. The first packet was sent to the receiver and all other duplicate ones are dropped.

The last test verifies ICMP packets.

|  |  |  |
| --- | --- | --- |
| ICMP | Interfaces | counted packets |
| Switch #2 | Eth 1 | 1 |
| Eth 11 | 50 |
| Eth 13 | 49 |

Table 13: Results of the c application verification with ICMP packets

The verification test with ICMP shows as expected the correct function of the duplicate packet detection.

## Automated verification

For automatization of the whole validation process of the duplicate packet detection, a script was implemented. This script ensures automatization by testing the java and the c application. It is able to set parameters on an additional configuration file. With this file, it is very flexible to setup the test to verify the correctness of the software development. In chapter B.3, the script inclusive configuration file is shown. The script is able to open SSH connections to all devices in the test arrangement, which is shown in chapter 3.1. To setup the SSH connections correctly, there are some commands to do at first. It is necessary to share RSA keys with all needed devices in the testbed, because all devices are password protected. After a SSH connection is created, the script starts capturing with tcpdump on the sender, the receiver and the switch #2. The next step is to load the configuration file, which is needed bring flexibility into the script. After that, there is a choice, if the c application or the java application should be used, which can be set in the configuration file. In addition, to set the interface where the application should sniff, send and drop can be chosen in the configuration file.

Furthermore, optionally the OpenFlow rules will be set again to prevent a failed test. For the verification of the c and the java application, a scapy test will be started which send packets to the receiver. The scapy script is based on python and is shown in chapter B.3.3. With Scapy, it is possible to create every packet modular. For the measurement of the data rate, the packet loss and the delay, a software called patheval is used. Patheval is a measurement tool, which can be started with python scripts. The duration of the test is adjustable in the configuration file of patheval. The used patheval scripts are shown in chapter B.3.4 . When the test is finished, the script open again SSH terminals in which the captured data are copied to switch #2. The reason therefore is to collect all captured data at one place. The captured files are marked with timestamps, so the files can be recognized to each test by comparing the dates by each other. Next to the last step, all used OpenFlow rules will be deleted. In the last steps, all captured data, which are not on switch #2 will be deleted to prevent useless memory wastage on all the other devices.

## Methodology of measurements

For the measurement, PathEval was used. With Patheval, it is possible to send UDP packets through the test arrangement. It is also possible to set the transmission rate. For a correct delay measurement, the sender and the receiver are synchronized to a NTP server[9]. Each measurement will last one minute and the interval for sending a packet will be set to 10us. For a good comparison, the measurement was made with five scenarios, which will be shown in the next Table.

|  |  |  |
| --- | --- | --- |
| **Measurement Scenario** | **Payload in Byte** | **Sended data rate in Mbit/s** |
| **Scenario A** | 600 | 64,77 |
| **Scenario B** | 800 | 85,73 |
| **Scenario C** | 1000 | 106,13 |
| **Scenario D** | 1100 | 116,70 |
| **Scenario E** | 1250 | 133,21 |

Table 14: Methodology of measurements

As shown in Table 14, five measurements are performed with Scenario A: 600 Byte/Packet, Scenario B: 800 Byte/Packet, Scenario C: 1000 Byte/Packet, Scenario D: 1100 Byte/Packet and Scenario E: 1250 Byte/Packet. With each measurement, the transmission rate is set to 10us per packet.

In each scenario, the delay from sender to the receiver and the received packets was captured.

# Results

In this section, the performance of the high redundant concept is shown. As mentioned in chapter 3, the duplicate packet detection in java and c version has to be tested. The duration of the measurement is about ten seconds. In addition, the delay from the sender to the receiver will be measured. A shell script was developed to test both applications as a local process, which is described in chapter 3.6. To prepare the figures, a MATLAB script is used. For a better view, the following shown figures are zoomed between second 1 and second 2. The y-axis on the left side of each figure will be the scale for the delay/packet in milliseconds and the y-axis on the right side of the measurement figures will scale the lost packets per interval. The x-axis will represent the time during the measurement.

### Measuring the Java application

C:\Users\wiela\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Unbenannt.jpg

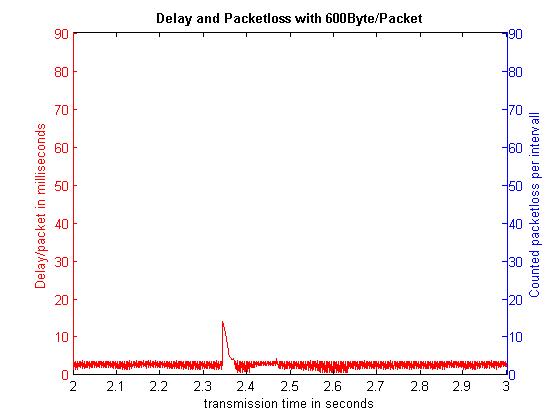


Figure 9: Scenario A with Java Application

The first measurement was done with a payload from 600 Bytes /packet. The calculated mean delay in this scenario is about 2,81ms. In scenario A, there was no packet loss detected. The data rate results in this case 64,77 Mbit/s.

C:\Users\wiela\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Unbenannt.jpg

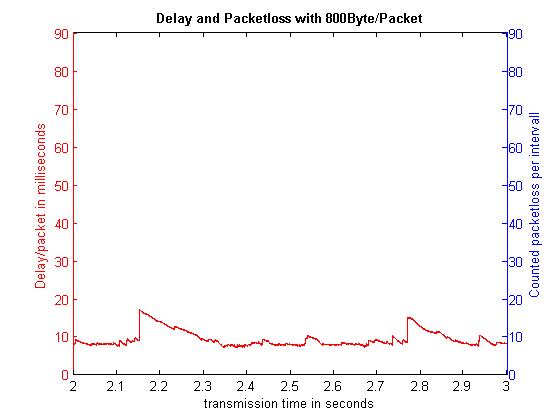


Figure 10: Scenario B with Java Application

In this Case, the packets were sent with 800 Bytes payload /packet. As seen in Figure 10, the average delay with 800 Bytes/Packet is 9,19ms. Furthermore, in scenario B there was no packet loss detected. The data rate in this scenario is 85,73 Mbit/s.

C:\Users\wiela\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Unbenannt.jpg

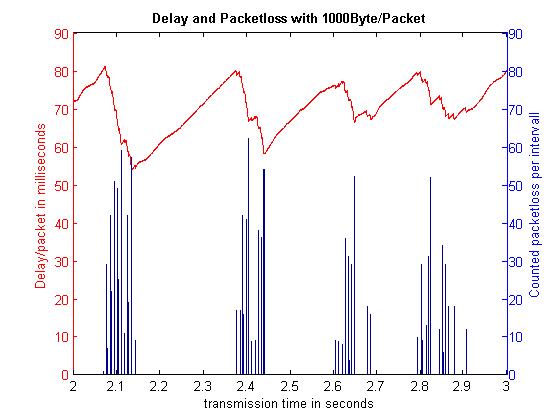


Figure 11: Scenario C with Java Application

The measurement with 1000 Byte payload /packet shows a first stress test for the java application with an average delay of 65,55ms.

As seen in Figure 11, there is a queuing behaviour, because of the set puffer in the java application. If the puffer is full, the application cannot handle the speed of received packets and lose packets. This is the first measurement with a packet loss by 11,06%. Scenario C reaches a data rate of 94,38Mbit/s. In comparison to the sended data rate in Scenario C, the reached data rate is 11,75Mbit/s slower than it should be.

C:\Users\wiela\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Unbenannt.jpg

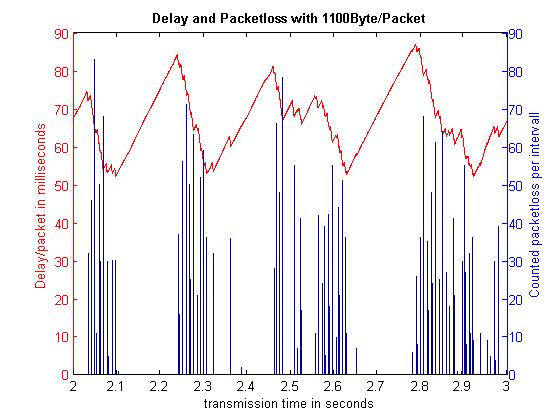


Figure 12: Scenario D with Java Application

In the fourth test scenario of the java application, the average delay is 65,28ms. In addition there is shown, that the stress test of the application will be harder because of the increased packet loss.

In Comparison to scenario C, scenario D has nearly the same data rate, but scenario D has more packet loss. In this case, the packet loss was increased to 18,72%. Considering to the sended data rate, the result is 22,32Mbit/s slower.

C:\Users\wiela\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Unbenannt.jpg

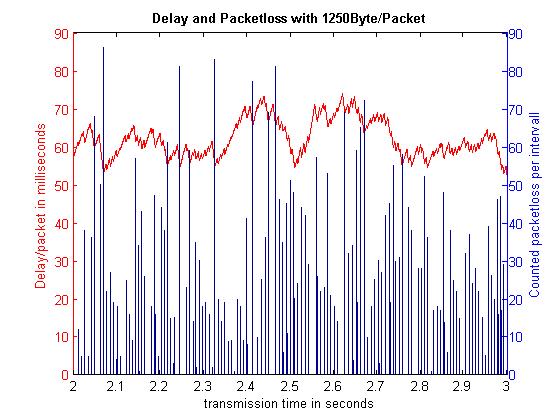


Figure 13: Scenario E with Java Application

In the last scenario, the average delay of the packets is 60,36ms. With the increased payload to 1250 Byte/Packet, the packet loss is also increased. In scenario E, the percentage packet loss was again increased to 28,38%. Because of the increased packet loss, the data rate is still nearly the same as in scenario C and D. With these results, the transmission is 38,09Mbit/s slower, than it should be.

In comparison to all scenarios, it seems that the duplicate packet detection with the java version can maximal transport around 100 Mbit/s.

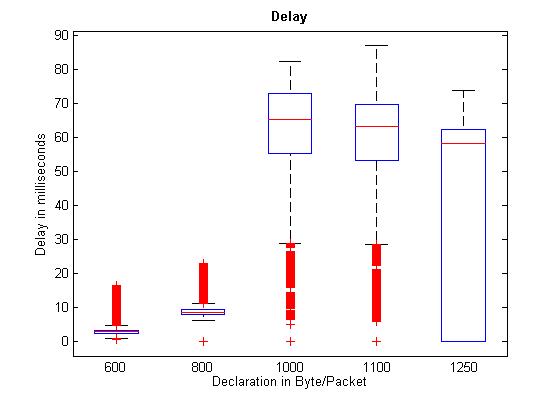


Figure 14: Comparison of delay from all measurements with Java application

In

Figure 14 is shown a comparison of the delay of every scenario. On the first Boxplot that represents scenario A, it is able to see that the maximum delay is only 4ms and the minimum delay is nearly 0ms. Only peaks were there with a maximum of 17ms. The second Boxplot, which represents the measurement with the packet payload of 800 Byte/Packet, shows, that in the second scenario the delay is minimal increased. The delay peak in this case is 23ms. The third scenario needs much more time for delivering the packets.

As seen in Figure 11, with 1000 Bytes/Packet, the application needs more time to capture the payload and looking for a duplicate one, so this could be an explanation for the delay time. In this case, the maximum delay time is 80ms. Scenario D is nearly the same as Scenario C, with the difference, that the maximum delay time was increased to 85ms. In addition, this can be explained with the puffer. The last Scenario shows a big range of needed delay time.

A suspicion is that the delay time does not need so much time than in Scenario D and C because of the high packet loss.

### Measuring the C application

C:\Users\wiela\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Unbenannt.jpg

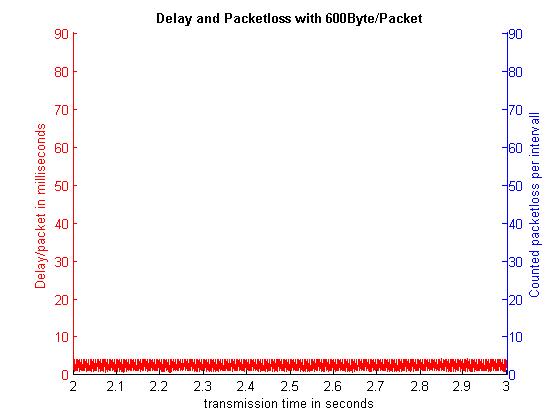


Figure 15: Scenario A with C Application

In scenario A, the mean delay is 2,42ms without any packet loss. The data rate in this measurement is 64,77 Mbit/s. In comparison to the java version of the duplicate packet detection, this version is nearly twice as fast.

C:\Users\wiela\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Unbenannt.jpg

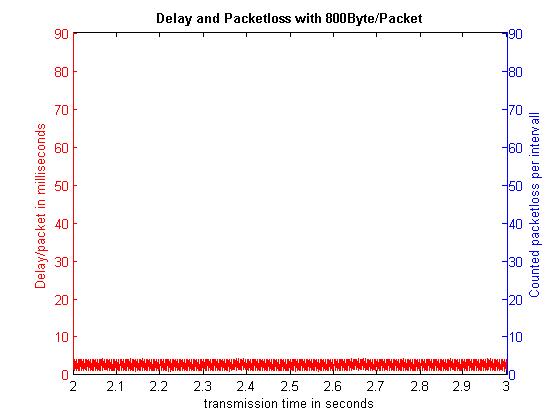


Figure 16: Scenario B with C Application

In Figure 16, the mean delay with 800 Bytes/packet is 2,59ms. As in scenario A, there was no packet loss detected. The data rate in this scenario is 85,73 Mbit/s.

C:\Users\wiela\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Unbenannt.jpgC:\Users\wiela\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Unbenannt.jpg

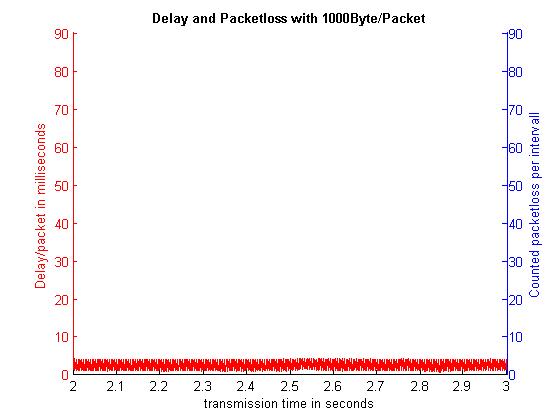


Figure 17: Scenario C with C Application

C:\Users\wiela\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Unbenannt.jpgScenario C shows, as the scenarios before, no complication with sorting out the duplicate packets. This measurement with 1000 Byte/Packet, the average delay is still 2,49ms. In comparison to the test with the java application in Figure 11, there is no queuing behaviour. In addition, there was still no packet loss detected. In this case, the data rate is 106,13 Mbit/s.

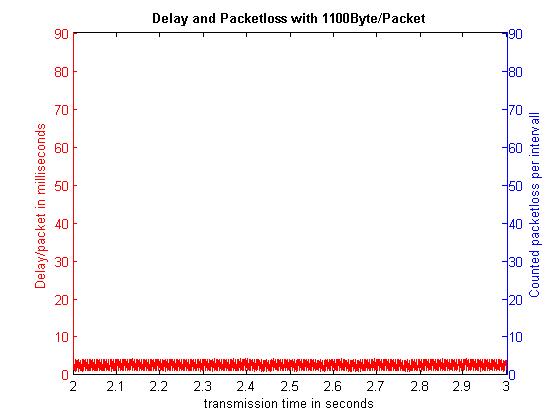


Figure 18: Scenario D with C Application

Figure 18 shows scenario D of the measurement. The mean delay is also by a payload about 1100 Bytes/packet 2,50ms. Packet loss does also not exist.

C:\Users\wiela\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Unbenannt.jpgIn scenario D, the resulting data rate is 116,70 Mbit/s.

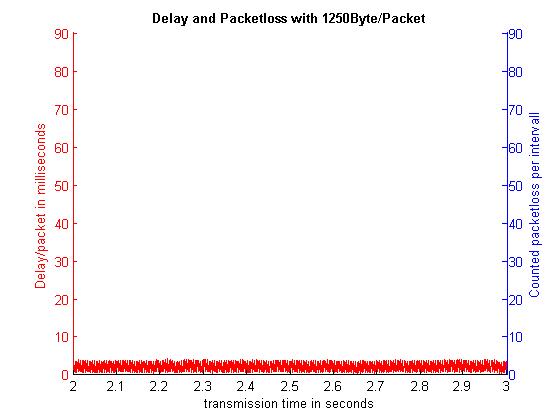


Figure 19: Scenario E with C Application

In the last scenario, the average delay of the packets is unchanged and very low with 2,05ms. With the increased payload to 1250 Byte/Packet, the packet loss is in comparison to the tested java application still missing. The resulting data rate is 133,21 Mbit/s. The received data rate of the measurements with the c application is the same as the sended data rate.

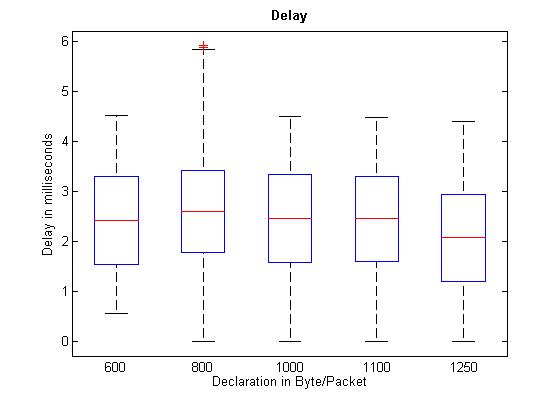


Figure 20: Comparison of delay from all measurements with C application

In Figure 20 is shown a comparison of the delay of every scenario as in

Figure 14. For a better view, the scale of the y-axis was set to six. There is to say, that the average delay is in all scenarios very low between 1,5ms and 3ms. There are still peaks 6ms, but in comparison to the tested java application, it is much better.

After the measurement, another measuring was taken. The goal of this measurement was to find out, at which data rate, packet loss occurs. The results shows, that the duplicate packet detection with the c application can reach a data rate under constant load up to 450Mbit/s. By increasing the data rate, first packet loss occurs.

# Conclusion

In consideration of the given goal, the duplicate packet detection with the c version has enforced. In comparison to the java version, the c version does not have any packet loss and can reach more data throughput than the java version. The data rate by using the c version can increased to 450Mbit/s without packet loss. After increasing the data rate, first packet loss occur.

There is to say that none of the versions can keep up with the speed of the original HSR protocol. Reason therefore could be the used method to compare the whole payload for detecting duplicate packets. Furthermore, there is to say, that both applications have potential to increase performance by editing the code.

One way would be to integrate the applications directly on network adapters with OpenOnload or DPDK.

In next steps, there could be improved the applications by using libraries for puffer the payload or executing the applications with special network adapter like OpenOnload or DPDK. In this thesis, the packets will be sent from two interfaces via OpenFlow rule to one virtual interface. The applications sniff the packets from this virtual interface and send it to the receiver or drop it. With OpenOnload, the applications would run directly on one supported interface. If the applications run on one OpenOnload interface, the second interface would be ignored. Now it is not possible with these applications to run with a proper function on an interface with OpenOnload.

To solve this problem, the applications can be extended to sniff parallel from two interfaces at the same time by using multithreading. This option is not implemented yet. The same problem is with running the applications with DPDK directly on an Intel network adapter.

# Bibliography

|  |  |
| --- | --- |
| [1] | D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky and S. Uhlig, “Software-Defined Networking: A Comprehensive Survey,” in *Proceedings of the IEEE, vol. 103, no. 1*, January 2015, pp. 14-76. |
| [2] | H. Kim and N. Feamster, “Improving network management with software defined networking,” in *Communications Magazine, IEEE , vol.51, no.2*, February 2013, pp. 114 - 119. |
| [3] | M. Jarschel, T. Zinner, T. Hossfeld, P. Tran-Gia and W. Kellerer, “Interfaces, attributes, and use cases: A compass for SDN,” in *IEEE Communications Magazine, vol. 52, no. 6*, June 2014, pp. 210-217. |
| [4] | C. Wieland and L. Steiner, “Evaluierung von Realtime Ethernet Protokollen,” Februar 2016. |
| [5] | H. Heine and O. Kleineberg, “The High-Availability Seamless redundancy protocol (HSR): Robust faulttolerant,” in *Factory Communication Systems (WFCS), 2012 9th IEEE International Workshop on*, Lemgo, 2012, pp. 213-222. |
| [6] | Y. Lin, D. Pitt, D. Hausheer, E. Johnson and Y. B. Lin, “Software-Defined Networking: Standardization for Cloud Computing's Second Wave,” in *Computer, vol. 47, no. 11*, Nov. 2014, pp. 19-22. |
| [7] | N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker and J. Turner, “OpenFlow: Enabling Innovation in Campus Networks,” 2008. [Online]. Available: http://archive.openflow.org/documents/openflow-wp-latest.pdf. [Accessed 09 02 2016]. |
| [8] | S. Sezer, S. Scott-Hayward, P. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller and N. Rao, “Are we ready for SDN? Implementation challenges for software-defined networks,” in *Communications Magazine, IEEE ,vol.51, no.7*, 2013, pp. 36 - 43. |
| [9] | D. Mills, “RFC 958: Network Time protocol (NTP),” September 1985. |

# Attachment

## Java application to detect duplicated packets

import org.jnetpcap.Pcap;

import org.jnetpcap.PcapIf;

import org.jnetpcap.packet.PcapPacket;

import org.jnetpcap.packet.PcapPacketHandler;

import org.jnetpcap.protocol.lan.Ethernet;

import java.io.\*;

import java.util.ArrayList;

import java.util.Date;

import java.util.List;

import java.util.Scanner;

/\*\*

\* detection.java

\* Purpose: Looking for duplicate packets and send them to the specific Interfaces

\* @author Christopher Wieland

\* @version 1.0 08/02/16

\*/

class detection {

private static final String log\_datei = "log.txt";

private static void log(String txt) {

Date dat = new Date();

try(PrintWriter out = new PrintWriter(new BufferedWriter(new FileWriter(log\_datei, true)))) {

out.println("[" + dat.toString() + "] " + "Payload of sended packet: " + " - " + txt);

}catch (IOException e) {

e.printStackTrace();

}

}

public static void main(String[] args) {

/\*\*

\* Variables and Lists

\*/

final Integer buffersize = 10000; // Set the length of the Array, which saves the Hash Codes, save the data 5 seconds by 2Packets/1ms

final List<String> payloadbuffer = new ArrayList<>(buffersize); // Will be filled with the Hashes of the packets

List<PcapIf> alldevices = new ArrayList<>(); // Will be filled with NICs

StringBuilder errbuf = new StringBuilder(); // For any error msgs

Scanner consoleinput = new Scanner(System.in);

final Ethernet eth = new Ethernet();

/\*\*

\* Find all Interfaces

\*/

Pcap.findAllDevs(alldevices, errbuf); // To find all possible Interfaces to sniff

/\*\*

\* Show all Interfaces

\*/

System.out.println("Network devices found:");

int NICs = 0;

for (PcapIf device : alldevices) {

String description = (device.getDescription() != null) ? device.getDescription(): "No description available";

System.out.printf("#%d: %s [%s]\n", NICs++,device.getName() ,description);

}

/\*\*

\* Decision, from which interface you will get the packets and send the packets

\*/

System.out.printf("\nChoose a sniffdevice!\n"); //Set a device to sniff

int choosedevicesniff = consoleinput.nextInt();

System.out.printf("\nChoose a senddevice!\n"); //Set a device to send

int choosedevicesend = consoleinput.nextInt();

System.out.printf("\nDo you want to log the payload? '0' - no, '1' - yes\n"); //Question for logging the payloadstring

final int logging = consoleinput.nextInt();

PcapIf getdevice = alldevices.get(choosedevicesniff);

PcapIf senddevice = alldevices.get(choosedevicesend);

System.out.printf("\n '%s' is chosen to get packet\n",(getdevice.getDescription() != null)?getdevice.getDescription(): getdevice.getName());

System.out.printf("\n '%s' is chosen to send packet\n",(senddevice.getDescription() != null)?senddevice.getDescription(): senddevice.getName());

/\*\*

\* Set the parameters for opening the port of the Interface from where you get the packets

\*/

int snaplen = 64 \* 1024; // maximum number of Bytes per packet

int flags = Pcap.MODE\_PROMISCUOUS; // capture all packets

int timeout = 0; // no timeout

/\*\*

\* Open Ports of the Interfaces

\*/

final Pcap sniff\_if = Pcap.openLive(getdevice.getName(), snaplen, flags, timeout, errbuf); //Open channel to capture packet from inputdevice

final Pcap send\_if = Pcap.openLive(senddevice.getName(), snaplen, flags, timeout, errbuf); //Open channel to send packet to destination port

/\*\*

\* Function for capturing the new packet

\* @param listiterator

\* @param payload

\* @param payloadstring

\*/

PcapPacketHandler<String> jpacketHandler = new PcapPacketHandler<String>() {

int listiterator = 0;

byte[] payload;

String payloadstring;

public void nextPacket(PcapPacket packet, String user) { //Sniff the packets

if (packet.hasHeader(eth)) { //to get the payload of the packet

payload = eth.getPayload();

try {

payloadstring = new String(payload, "UTF-8"); //cast the byte array to string for output

} catch (UnsupportedEncodingException e) {

e.printStackTrace();

}

}

if(payloadbuffer.size() < buffersize && !payloadbuffer.contains(payloadstring)) { //If the Array isn't full and the Payload doesn't exist in the Array, add a packet in the array

payloadbuffer.add(payloadstring); //Add Payload of the packet in the Array

if(logging == 1) log(payloadstring); //Put Payload of the sended packet in the file

send\_if.sendPacket(packet); //send packet to output Port

}

else if(payloadbuffer.size() == buffersize && !payloadbuffer.contains(payloadstring)){ //If the Array have already packets in the list and the Payload doesn't exist in the Array, replace them with new

payloadbuffer.set(listiterator, payloadstring); //replace the packet with an older one

listiterator++;

listiterator %= buffersize; //Iteration only as long as the Array is, then begin by zero again

if(logging == 1) log(payloadstring); //Put Payload of the sended packet in the file

send\_if.sendPacket(packet);

}

}

};

/\*\*

\* Loop to capture continiual packets from the sniffing Interface

\*/

sniff\_if.loop(Pcap.LOOP\_INFINITE, jpacketHandler, ""); //to capture the Packets continiual

}

}

Script 3: Java Application to detect duplicate packets

## C application to detect duplicated packets

#include<pcap.h>

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<net/ethernet.h>

#include<net/if.h>

#include<netinet/ip.h>

#include<netinet/udp.h>

#include<netinet/ether.h>

#include<netinet/in.h>

#include<sys/socket.h>

#include<arpa/inet.h>

#include<linux/if\_packet.h>

#include<linux/sockios.h>

struct \_ListItem{

char payload[20000];

struct \_ListItem \* prev;

struct \_ListItem \* next;

};

typedef struct \_ListItem LISTITEM;

struct \_List{

struct \_ListItem \* first;

struct \_ListItem \* last;

unsigned count;

};

typedef struct \_List LIST;

const char \*SNIFFDEVICE, \*SENDDEVICE, \*DROPDEVICE;

char \*SENDDEVMAC0, \*SENDDEVMAC1, \*SENDDEVMAC2, \*SENDDEVMAC3, \*SENDDEVMAC4, \*SENDDEVMAC5;

char \*DROPDEVMAC0, \*DROPDEVMAC1, \*DROPDEVMAC2, \*DROPDEVMAC3, \*DROPDEVMAC4, \*DROPDEVMAC5;

FILE \*logfile;

int itemcounter = 0, sending, dropping, BUFFERSIZE, LOGGING;

LIST lst;

LISTITEM\* newItem;

struct ifreq if\_idx;

struct sockaddr\_ll socket\_address;

//Function prototypes

void process\_packet(u\_char \*, const struct pcap\_pkthdr \*, const u\_char \*);

void PrintData (const u\_char \*, long, char []);

void WriteData (const u\_char \*, long, char []);

int appendint(int a, int b);

LIST\* InitList(LIST\*);

LIST\* CreateNewList();

LISTITEM\* InitListItem(LISTITEM\*);

LISTITEM\* CreateNewListItem();

void AppendItemToList(LIST\*, LISTITEM\*);

void WriteStreamtoItem(LISTITEM\* item, char [], long);

int SearchItemInList(LIST\* lst, char [], char[], long);

void DeleteItemFromList(LIST\*);

int comparearray(char[], char[], long);

int main(int argc, char \*argv[]){

pcap\_if\_t \*alldevices , \*device;

pcap\_t \*sniffinterface;

char errbuf[100], devices[100][100];

unsigned int count = 1;

printf("\n");

printf("/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("/\* \*\n");

printf("/\* Author: Christopher Wieland \*\n");

printf("/\* Name: Duplicate Packet Detection \*\n");

printf("/\* Company: Salzburg Research Forschungsgesellschaft mbH \*\n");

printf("/\* Datum: April, 2016 \*\n");

printf("/\* Copyright: All rights reserved \*\n");

printf("/\* \*\n");

printf("/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

//Set the parameters for the program

BUFFERSIZE = atoi(argv[1]);

LOGGING = (int)(argv[2][0] - '0');

SNIFFDEVICE = (argv[3]);

SENDDEVICE = (argv[4]);

DROPDEVICE = (argv[5]);

SENDDEVMAC0 = (argv[6]);

SENDDEVMAC1 = (argv[7]);

SENDDEVMAC2 = (argv[8]);

SENDDEVMAC3 = (argv[9]);

SENDDEVMAC4 = (argv[10]);

SENDDEVMAC5 = (argv[11]);

DROPDEVMAC0 = (argv[12]);

DROPDEVMAC1 = (argv[13]);

DROPDEVMAC2 = (argv[14]);

DROPDEVMAC3 = (argv[15]);

DROPDEVMAC4 = (argv[16]);

DROPDEVMAC5 = (argv[17]);

//Create a new Bufferlist for captured payloaddata

CreateNewList();

//Get the list of available devices

if( pcap\_findalldevs( &alldevices , errbuf) )

{

printf("Error finding devices : %s" , errbuf);

exit(1);

}

//Open the device for sniffing

sniffinterface = pcap\_open\_live(SNIFFDEVICE , 65536 , 1 , 0 , errbuf);

//create a logfile, if the user want to log all data

if(LOGGING == 1){

logfile=fopen("dpd\_log\_payload.txt","w");

if(logfile==NULL)

{

printf("Unable to create file.");

}

}

//Put the device in capture loop

pcap\_loop(sniffinterface , -1 , process\_packet , NULL);

return 0;

}

void process\_packet(u\_char \*args, const struct pcap\_pkthdr \*header, const u\_char \*buffer){

long size = header->len;

char dataload[size];

char sucharray[size];

int i;

//printf("NEW PACKET #%d\n", itemcounter);

//Zero all buffer

memset(dataload, 0, size);

memset(sucharray, 0, size);

// Use the function PrintData for logging all data or the function

// WriteData for only do the job

if (LOGGING == 1){

fprintf(logfile , "\n\n###########################################################\n");

PrintData(buffer + sizeof(struct ethhdr) , size - sizeof(struct ethhdr), dataload );

fprintf(logfile , "\n\n###########################################################\n");

}

else{

WriteData(buffer + sizeof(struct ethhdr) , size - sizeof(struct ethhdr), dataload );

}

//Create a new Item and write the captured payload in it

newItem = CreateNewListItem();

WriteStreamtoItem(newItem, dataload, size);

//Variable to compare the new payload from the payload in the Bufferlist

SearchItemInList(&lst, dataload, sucharray, size);

/\*

printf("LOAD : ");

for(i=0 ; i < size ; i++){

printf("%c", dataload[i]);

}

printf("\n");

printf("SUCHARRAY: ");

for(i=0 ; i < size ; i++){

printf("%c", sucharray[i]);

}

printf("\n");

\*/

//If the new payload isn't in the Bufferlist yet, then forward the

//packet and add the new payload to the Bufferlist

if ((comparearray(dataload, sucharray, size) == 0) && itemcounter >0 ){

//printf("SENDING\n\n\n");

//Open the device for sending

sending = socket(AF\_PACKET, SOCK\_RAW, IPPROTO\_RAW);

//Get index of the sending interface

memset(&if\_idx, 0, sizeof(struct ifreq));

strncpy(if\_idx.ifr\_name, SENDDEVICE, IFNAMSIZ-1);

if (ioctl(sending, SIOCGIFINDEX, &if\_idx) < 0)

perror("SIOCGIFINDEX");

//Append the new payload to the Bufferlist

AppendItemToList(&lst, newItem);

// Index of the sending device

socket\_address.sll\_ifindex = if\_idx.ifr\_ifindex;

// Set the Address length

socket\_address.sll\_halen = ETH\_ALEN;

// Set the Destination MAC, where the packet sould be forwarded

socket\_address.sll\_addr[0] = (long)SENDDEVMAC0;

socket\_address.sll\_addr[1] = (long)SENDDEVMAC1;

socket\_address.sll\_addr[2] = (long)SENDDEVMAC2;

socket\_address.sll\_addr[3] = (long)SENDDEVMAC3;

socket\_address.sll\_addr[4] = (long)SENDDEVMAC4;

socket\_address.sll\_addr[5] = (long)SENDDEVMAC5;

// Send packet

if (sendto(sending, buffer, size, 0, (struct sockaddr\*)&socket\_address, sizeof(struct sockaddr\_ll)) < 0) printf("Send failed\n");

close(sending);

//Count the itemcounter for the deletefunction

itemcounter++;

}

//If the payload is already in the Bufferlist,

// drop it

if ((comparearray(dataload, sucharray, size) == 1) && itemcounter >0){

//printf("DROPPING\n\n\n");

//Open the device for dropping

dropping = socket(AF\_PACKET, SOCK\_RAW, IPPROTO\_RAW);

//Get index of the dropping interface

memset(&if\_idx, 0, sizeof(struct ifreq));

strncpy(if\_idx.ifr\_name, DROPDEVICE, IFNAMSIZ-1);

if (ioctl(dropping, SIOCGIFINDEX, &if\_idx) < 0)

perror("SIOCGIFINDEX");

// Index of the dropping device

socket\_address.sll\_ifindex = if\_idx.ifr\_ifindex;

// Set the Address length

socket\_address.sll\_halen = ETH\_ALEN;

// Set the Destination MAC, where the packet sould be forwarded

socket\_address.sll\_addr[0] = (long)DROPDEVMAC0;

socket\_address.sll\_addr[1] = (long)DROPDEVMAC1;

socket\_address.sll\_addr[2] = (long)DROPDEVMAC2;

socket\_address.sll\_addr[3] = (long)DROPDEVMAC3;

socket\_address.sll\_addr[4] = (long)DROPDEVMAC4;

socket\_address.sll\_addr[5] = (long)DROPDEVMAC5;

// Drop packet

if (sendto(dropping, buffer, size, 0, (struct sockaddr\*)&socket\_address, sizeof(struct sockaddr\_ll)) < 0) printf("Drop failed\n");

close(dropping);

}

//If the first packet will be captured, add the payload to the

// Bufferlist and forward the packet to the destination

if(itemcounter == 0){

//printf("SENDING\n\n\n");

//Open the device for sending

sending = socket(AF\_PACKET, SOCK\_RAW, IPPROTO\_RAW);

//Get index of the sending interface

memset(&if\_idx, 0, sizeof(struct ifreq));

strncpy(if\_idx.ifr\_name, SENDDEVICE, IFNAMSIZ-1);

if (ioctl(sending, SIOCGIFINDEX, &if\_idx) < 0)

perror("SIOCGIFINDEX");

AppendItemToList(&lst, newItem);

// Index of the sending device

socket\_address.sll\_ifindex = if\_idx.ifr\_ifindex;

// Set the Address length

socket\_address.sll\_halen = ETH\_ALEN;

// Set the Destination MAC, where the packet sould be forwarded

socket\_address.sll\_addr[0] = (long)SENDDEVMAC0;

socket\_address.sll\_addr[1] = (long)SENDDEVMAC1;

socket\_address.sll\_addr[2] = (long)SENDDEVMAC2;

socket\_address.sll\_addr[3] = (long)SENDDEVMAC3;

socket\_address.sll\_addr[4] = (long)SENDDEVMAC4;

socket\_address.sll\_addr[5] = (long)SENDDEVMAC5;

// Send packet

if (sendto(sending, buffer, size, 0, (struct sockaddr\*)&socket\_address, sizeof(struct sockaddr\_ll)) < 0) printf("Send failed\n");

close(sending);

//Count the itemcounter for the deletefunction

itemcounter++;

}

//Delete the first item in the Buffer list after the set count

if (itemcounter >= BUFFERSIZE){

DeleteItemFromList(&lst);

//Subtract the itemcounter by one for the new packet

itemcounter --;

}

}

LIST\* InitList(LIST\* list){

//Initialize the whole list

if( list == NULL )

return NULL;

list->first = list->last = NULL;

list->count = 0;

return list;

}

LIST\* CreateNewList(){

//Get space for the new List

LIST\* list = (LIST\*)malloc(sizeof(LIST));

if( list == NULL )

return NULL;

//Initialize the new List

InitList(list);

return list;

}

LISTITEM\* CreateNewListItem(){

//Get space for a new item

LISTITEM\* item = (LISTITEM\*)malloc(sizeof(LISTITEM));

if( item == NULL )

return NULL;

//Initialize the new Item

InitListItem(item);

return item;

}

LISTITEM\* InitListItem(LISTITEM\* item){

//Initialize the Item

if( item == NULL )

return NULL;

item->prev = item->next = NULL;

return item;

}

void AppendItemToList(LIST\* lst, LISTITEM\* item){

//If the item is the first, set first and last pointer to this

if( lst->count == 0 )

{

lst->first = lst->last = item;

item->prev = item->next = NULL;

}

//else append the new Item on the oldest item in the Bufferlist

// and set the last pointer on it

else

{

lst->last->next = item;

item->prev = lst->last;

item->next = NULL;

lst->last = item;

}

//Count the list by one for each new packet

lst->count++;

}

int SearchItemInList(LIST\* lst, char data[], char such[], long size) {

int i;

//Create a temporary listitem\* and set it on the first pointer

LISTITEM\* tmpitem = lst->first;

//While the temporary item isn't NULL, compare the payload of it

//with the data

while(tmpitem !=NULL){

if (comparearray(tmpitem->payload,data, size) == 1){

for(i=0 ; i < size; i++){

sprintf(&such[i], "%c", tmpitem->payload[i]);

}

return 1;

}

tmpitem = tmpitem ->next;

}

if(tmpitem ==NULL){

return 0;

}

}

void DeleteItemFromList(LIST\* lst){

//Create a temporary listitem\* and set it on the first pointer

LISTITEM\* help = lst->first;

if( lst == NULL || lst->count == 0)

return;

//Delete the first item in the Bufferlist

lst->first = help->next;

free(help);

//Subtract the counter of the list by one

lst->count--;

}

void WriteStreamtoItem(LISTITEM\* item, char dataload[], long size){

//Write the payload of the captured packet to the newItem

int i;

for(i=0 ; i < size ; i++){

sprintf(&item->payload[i], "%c", dataload[i]);

}

}

void PrintData (const u\_char \* data , long Size, char load[]){

int i , j, k;

for(i=0 ; i < Size ; i++)

{

if( i!=0 && i%16==0) //If the first line of Hex is finished.

{

fprintf(logfile , " ");

fprintf(logfile , "\n");

}

if(i%16==0) fprintf(logfile , " ");

//Copy all data from the captured payload to the

//buffer, which is made to send the new packet

for(k=0 ; k < Size ; k++){

//Copy all data from the captured payload to the buffer

if ( isprint(data[k]) ){ /\* If it is a printable character, print it \*/

sprintf(&load[k], "%c", data[k]);

}

}

//Write the payload into the created logfile

fprintf(logfile , " %02X",(unsigned int)load[i]);

//Just for a better view

if( i==Size-1)

{

for(j=0;j<15-i%16;j++)

{

fprintf(logfile , " ");

}

fprintf(logfile , " ");

fprintf(logfile , "\n" );

}

}

}

void WriteData (const u\_char \* data , long Size, char load[]){

int i;

for(i=0 ; i < Size ; i++){

//Copy all data from the captured payload to the buffer

if ( isprint(data[i]) ){

sprintf(&load[i], "%c", data[i]);

}

}

}

int comparearray(char array1[], char array2[], long size){

int i;

for(i=0; i<size; ++i){

if(array1[i]!=array2[i]){

//Es wurde nichts gefunden -> SEND

return 0;

}

}

//Es wurde etwas gefunden -> DROP

return 1;

}

Script 4: C Application to detect duplicate packets

## Script for test automatization

### Shell

#!/bin/bash

###############################

#open capturing for interfaces#

###############################

gnome-terminal -x bash -c "ssh 110 'timeout $timeout+15 tcpdump -i eth13 -w DUP\_Test\_capture/DUP\_Test\_capture\_110/['$(date +%Y%m%d\_%H:%M:%S)']verworfen\_110.pcap; timeout $timeout+16 exit'"

gnome-terminal -x bash -c "ssh 110 'timeout $timeout+15 tcpdump -i eth1 -w DUP\_Test\_capture/DUP\_Test\_capture\_110/['$(date +%Y%m%d\_%H:%M:%S)']weitergeleitet\_110.pcap; timeout $timeout+16 exit'"

gnome-terminal -x bash -c "ssh 110 'timeout $timeout+15 tcpdump -i eth11 -w DUP\_Test\_capture/DUP\_Test\_capture\_110/['$(date +%Y%m%d\_%H:%M:%S)']empfangen\_110.pcap; timeout $timeout+16 exit'"

gnome-terminal -x bash -c "ssh 115 'timeout $timeout+15 tcpdump -i eth1 -w DUP\_Test\_capture\_115/['$(date +%Y%m%d\_%H:%M:%S)']gesendet\_115.pcap; timeout $timeout+16 exit'"

gnome-terminal -x bash -c "ssh 119 'timeout $timeout+15 tcpdump -i p3p1 -w DUP\_Test\_capture\_119/['$(date +%Y%m%d\_%H:%M:%S)']empfangen\_119.pcap; timeout $timeout+16 exit'"

##################

#open config file#

##################

. /home/christopher/Schreibtisch/DUP\_config\_110.cfg

####################

#start code for DPD#

####################

if [ $javacode = 1 ] && [ $ccode = 0 ] ; then

#open Java program

gnome-terminal -x bash -c "ssh 110 'cd /root/packetdetection/Java/packetdetection/src/; javac -cp /root/packetdetection/Java/jnetpcap/jnetpcap.jar detection.java; java -cp .:/root/packetdetection/Java/jnetpcap/jnetpcap.jar detection $sniffdevice\_java $senddevice\_java $dropdevice\_java; timeout $timeout exit'"

fi

if [ $javacode = 0 ] && [ $ccode = 1 ] ; then

#open C program

gnome-terminal -x bash -c "ssh 110 'cd /root/packetdetection/C/; gcc -c main.c -static -lpcap; gcc -o main main.o -lpcap; ./main $Buffersize $Logging $sniffdevice\_c $senddevice\_c $dropdevice\_c $senddevicemac0 $senddevicemac1 $senddevicemac2 $senddevicemac3 $senddevicemac4 $senddevicemac5 $dropdevicemac0 $dropdevicemac1 $dropdevicemac2 $dropdevicemac3 $dropdevicemac4 $dropdevicemac5 ; timeout $timeout exit'"

fi

#########################

#do rule set on EX 121er#

#########################

gnome-terminal -x bash -c "ssh 110 'ovs-vsctl --db=tcp:192.168.40.121:6634 set-controller br0 ptcp:6633; ovs-ofctl add-flow tcp:192.168.40.121:6633 in\_port=6,actions=output:3; ovs-ofctl add-flow tcp:192.168.40.121:6633 in\_port=3,actions=output:5,6; ovs-vsctl --db=tcp:192.168.40.121:6634 del-controller br0'"

#########################

#pause before test begin#

#########################

sleep 4

###################

#start packet test#

###################

gnome-terminal -x bash -c "timeout $timeout ssh 115 'python DPD\_scapy.py; timeout $timeout+2 exit'"

#############################

#pause till the test is over#

#############################

sleep 15 #10 + timeout aus config File

###############################

#send captured data to main PC#

###############################

gnome-terminal -x bash -c "scp -3 -r root@192.168.40.119:/root/DUP\_Test\_capture\_119/ root@192.168.40.110:/root/DUP\_Test\_capture"

gnome-terminal -x bash -c "scp -3 -r root@192.168.40.115:/root/DUP\_Test\_capture\_115/ root@192.168.40.110:/root/DUP\_Test\_capture"

###################################

#Sure to quit everything on PC 110#

###################################a

gnome-terminal -x bash -c "ssh 110 'killall java; killall tcpdump; killall python'"

#########################

#delete rule on EX 121er#

#########################

gnome-terminal -x bash -c "ssh 110 'ovs-vsctl --db=tcp:192.168.40.121:6634 set-controller br0 tcp:192.168.40.104:6633 ptcp:6633; ovs-ofctl del-flows tcp:192.168.40.121:6633 in\_port=6; ovs-ofctl del-flows tcp:192.168.40.121:6633 in\_port=3'"

###########################

#pause before delete files#

###########################

sleep 5

##############################

#delete data from 115 and 119#

##############################

gnome-terminal -x bash -c "ssh 115 'rm /root/DUP\_Test\_capture\_115/\*'"

gnome-terminal -x bash -c "ssh 119 'rm /root/DUP\_Test\_capture\_119/\*'"

Script 5: Shell Script for test automatization

### Additional Configuration File

#####################

#Device List in Java#

#####################

#Network devices found:

#0: lo

#1: any [Pseudo-device that captures on all interfaces]

#2: eth13

#3: eth12

#4: eth11

#5: eth3

#6: eth2

#7: eth1

#8: nfqueue [Linux netfilter queue (NFQUEUE) interface]

#9: nflog [Linux netfilter log (NFLOG) interface]

#10: eth0

##################

#Device List in C#

##################

#Network devices found:

#1: eth0

#2: nflog [Linux netfilter log (NFLOG) interface]

#3: nfqueue [Linux netfilter queue (NFQUEUE) interface]

#4: eth1

#5: eth2

#6: eth3

#7: eth11

#8: eth12

#9: eth13

#10: any [Pseudo-device that captures on all interfaces]

#11: lo

##################################

#auswahl der richtigen interfaces#

##################################

#Deviceauswahl für die Java Applikation

sniffdevice\_java="4"

senddevice\_java="7"

dropdevice\_java="2"

#Deviceauswahl für die C Applikation

sniffdevice\_c="eth11"

senddevice\_c="eth1"

dropdevice\_c="eth13"

senddevicemac0="0xD4"

senddevicemac1="0x3D"

senddevicemac2="0x7E"

senddevicemac3="0xA8"

senddevicemac4="0xE4"

senddevicemac5="0xCC"

dropdevicemac0="0xD6"

dropdevicemac1="0x3B"

dropdevicemac2="0xC4"

dropdevicemac3="0xC1"

dropdevicemac4="0x7C"

dropdevicemac5="0x79"

#Weitere angaben für die C Applikation

Buffersize="10"

Logging="0"

######################################

#abfrage, welcher code verwendet wird#

######################################

#bei (1) wird der Code verwendet, bei (0) nicht

ccode="1"

javacode="0"

####################################

#abfrage, wie lange der Test dauert#

####################################

timeout="5"

Script 6: Additional Configuration File for Shell Script

### Scapy script

#!/usr/bin/env python

import sys

import logging

logging.getLogger("scapy.runtime").setLevel(logging.ERROR)

from scapy.all import \*

#send data with Layer2

#sendp("This is an Ethernet packet", iface="eth1", loop=1, inter=0.1)

#send data with UDP

#send(IP(dst="10.100.40.119")/UDP()/Raw(load="This is an Ethernet packet"), iface="eth1", loop=1, inter=0.1)

#send data with TCP

#send(IP(dst="10.100.40.119")/TCP()/Raw(load="This is an Ethernet packet"), iface="eth1", loop=1, inter=0.1)

#send data with ICMP

send(IP(dst="10.100.40.119")/ICMP()/Raw(load="This is an Ethernet packet"), iface="eth1", loop=1, inter=0.1)

Script 7: Scapy script to send data

### Patheval sender and receiver script

The sender script is the following:

#!/usr/bin/env python

import socket

TCP\_IP = '192.168.40.115'

TCP\_PORT = 45054

BUFFER\_SIZE = 1024

MESSAGE = "CMND=2\nTASK=1\nFLID=2000\nSEND=10.100.40.115\nRCPT=10.100.40.119\nENDT=10\nSRVP=20000\n"

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

s.connect((TCP\_IP, TCP\_PORT))

s.send(MESSAGE)

MESSAGE = "MPLX=1\nOPTS=255\nPROT=17\nTOSV=0\nAGRT=0\n"

s.send(MESSAGE)

MESSAGE = "PSDC=1\nPT01=1\nPP01=0\nP011=600\nSIDC=1\nST01=1\nSP01=0\nS011=5\nENDC\n"

s.send(MESSAGE)

data = s.recv(BUFFER\_SIZE)

print "received data:", data

MESSAGE = "\n\n"

s.send(MESSAGE)

data = s.recv(BUFFER\_SIZE)

s.close()

print "received data:", data

Script 8:Patheval sender script

The receiver script is the following:

#!/usr/bin/env python

import socket

TCP\_IP = '192.168.40.119'

TCP\_PORT = 45054

BUFFER\_SIZE = 1024

MESSAGE = "CMND=2\nTASK=0\nFLID=2000\nSEND=10.100.40.115\nRCPT=10.100.40.119\nENDT=10\nSRVP=20000\n"

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

s.connect((TCP\_IP, TCP\_PORT))

s.send(MESSAGE)

MESSAGE = "MPLX=1\nOPTS=255\nPROT=17\nTOSV=0\nAGRT=0\n"

s.send(MESSAGE)

MESSAGE = "PSDC=1\nPT01=1\nPP01=0\nP011=600\nSIDC=1\nST01=1\nSP01=0\nS011=5\nENDC\n"

s.send(MESSAGE)

data = s.recv(BUFFER\_SIZE)

print "received data:", data

MESSAGE = "\n\n"

s.send(MESSAGE)

data = s.recv(BUFFER\_SIZE)

s.close()

print "received data:", data

Script 9: Patheval receiver script

## Matlab Script for evaluation of the data

%%Messung der Paketdauer

%laden der Rohdaten

load Byte1250.dat

load Byte1100.dat

load Byte1000.dat

load Byte800.dat

load Byte600.dat

%Auswerten der Rohdaten

Byte1250x = sort(Byte1250(:,:,:,:));

sollpaket1250 = max(Byte1250x(:,1));

istpaket1250 = max(size(Byte1250x(:,1)));

Byte1100x = sort(Byte1100(:,:,:,:));

sollpaket1100 = max(Byte1100x(:,1));

istpaket1100 = max(size(Byte1100x(:,1)));

Byte1000x = sort(Byte1000(:,:,:,:));

sollpaket1000 = max(Byte1000x(:,1));

istpaket1000 = max(size(Byte1000x(:,1)));

Byte800x = sort(Byte800(:,:,:,:));

sollpaket800 = max(Byte800x(:,1));

istpaket800 = max(size(Byte800x(:,1)));

Byte600x = sort(Byte600(:,:,:,:));

sollpaket600 = max(Byte600x(:,1));

istpaket600 = max(size(Byte600x(:,1)));

%alle angekommenden pakete werden gezählt bei 1250 Byte payload

counter1250 = 1;

while (counter1250 ~= istpaket1250+1)

paketcounter1250(counter1250,1) = counter1250;

counter1250 = counter1250 + 1;

end

counterx1250 = 1;

while (counterx1250 ~= istpaket1250+1)

istpaketcounter1250(counterx1250,1) = Byte1250x(counterx1250,1);

counterx1250 = counterx1250 + 1;

end

%alle angekommenden pakete werden gezählt bei 1100 Byte payload

counter1100 = 1;

while (counter1100 ~= istpaket1100+1)

paketcounter1100(counter1100,1) = counter1100;

counter1100 = counter1100 + 1;

end

counterx1100 = 1;

while (counterx1100 ~= istpaket1100+1)

istpaketcounter1100(counterx1100,1) = Byte1100x(counterx1100,1);

counterx1100 = counterx1100 + 1;

end

%alle angekommenden pakete werden gezählt bei 1000 Byte payload

counter1000 = 1;

while (counter1000 ~= istpaket1000+1)

paketcounter1000(counter1000,1) = counter1000;

counter1000 = counter1000 + 1;

end

counterx1000 = 1;

while (counterx1000 ~= istpaket1000+1)

istpaketcounter1000(counterx1000,1) = Byte1000x(counterx1000,1);

counterx1000 = counterx1000 + 1;

end

%alle angekommenden pakete werden gezählt bei 800 Byte payload

counter800 = 1;

while (counter800 ~= istpaket800+1)

paketcounter800(counter800,1) = counter800;

counter800 = counter800 + 1;

end

counterx800 = 1;

while (counterx800 ~= istpaket800+1)

istpaketcounter800(counterx800,1) = Byte800x(counterx800,1);

counterx800 = counterx800 + 1;

end

%alle angekommenden pakete werden gezählt bei 600 Byte payload

counter600 = 1;

while (counter600 ~= istpaket600+1)

paketcounter600(counter600,1) = counter600;

counter600 = counter600 + 1;

end

counterx600 = 1;

while (counterx600 ~= istpaket600+1)

istpaketcounter600(counterx600,1) = Byte600x(counterx600,1);

counterx600 = counterx600 + 1;

end

%sendezeiten der Pakete bei 1250 Byte

startzeit1250 = Byte1250x(:,2);

endzeit1250 = Byte1250x(:,3);

%sendezeiten der Pakete bei 1100 Byte

startzeit1100 = Byte1100x(:,2);

endzeit1100 = Byte1100x(:,3);

%sendezeiten der Pakete bei 1000 Byte

startzeit1000 = Byte1000x(:,2);

endzeit1000 = Byte1000x(:,3);

%sendezeiten der Pakete bei 800 Byte

startzeit800 = Byte800x(:,2);

endzeit800 = Byte800x(:,3);

%sendezeiten der Pakete bei 600 Byte

startzeit600 = Byte600x(:,2);

endzeit600 = Byte600x(:,3);

%Berechnung des Delays in millisekunden bei 1250 byte

delay1250 = (endzeit1250(:,1) - startzeit1250(:,1))/1000;

%Berechnung des Delays in millisekunden bei 1100 byte

delay1100 = (endzeit1100(:,1) - startzeit1100(:,1))/1000;

%Berechnung des Delays in millisekunden bei 1000 byte

delay1000 = (endzeit1000(:,1) - startzeit1000(:,1))/1000;

%Berechnung des Delays in millisekunden bei 800 byte

delay800 = (endzeit800(:,1) - startzeit800(:,1))/1000;

%Berechnung des Delays in millisekunden bei 600 byte

delay600 = (endzeit600(:,1) - startzeit600(:,1))/1000;

%Berechnung der Paketverluste bei 1250 Byte

k1250 = 1;

losstemp1250 = 0;

while (k1250 ~= istpaket1250+1)

if(paketcounter1250(k1250,1)+losstemp1250 == istpaketcounter1250(k1250,1))

paketloss1250(k1250,1) = istpaketcounter1250(k1250,1) - (paketcounter1250(k1250,1)+losstemp1250);

end

if(paketcounter1250(k1250,1)+losstemp1250 ~= istpaketcounter1250(k1250,1))

paketloss1250(k1250,1) = istpaketcounter1250(k1250,1) - (paketcounter1250(k1250,1)+losstemp1250);

losstemp1250 = losstemp1250 + paketloss1250(k1250,1);

end

k1250 = k1250 + 1;

end

%Berechnung der Paketverluste bei 1100 Byte

k1100 = 1;

losstemp1100 = 0;

while (k1100 ~= istpaket1100+1)

if(paketcounter1100(k1100,1)+losstemp1100 == istpaketcounter1100(k1100,1))

paketloss1100(k1100,1) = istpaketcounter1100(k1100,1) - (paketcounter1100(k1100,1)+losstemp1100);

end

if(paketcounter1100(k1100,1)+losstemp1100 ~= istpaketcounter1100(k1100,1))

paketloss1100(k1100,1) = istpaketcounter1100(k1100,1) - (paketcounter1100(k1100,1)+losstemp1100);

losstemp1100 = losstemp1100 + paketloss1100(k1100,1);

end

k1100 = k1100 + 1;

end

%Berechnung der Paketverluste bei 1000 Byte

k1000 = 1;

losstemp1000 = 0;

while (k1000 ~= istpaket1000+1)

if(paketcounter1000(k1000,1)+losstemp1000 == istpaketcounter1000(k1000,1))

paketloss1000(k1000,1) = istpaketcounter1000(k1000,1) - (paketcounter1000(k1000,1)+losstemp1000);

end

if(paketcounter1000(k1000,1)+losstemp1000 ~= istpaketcounter1000(k1000,1))

paketloss1000(k1000,1) = istpaketcounter1000(k1000,1) - (paketcounter1000(k1000,1)+losstemp1000);

losstemp1000 = losstemp1000 + paketloss1000(k1000,1);

end

k1000 = k1000 + 1;

end

%Berechnung der Paketverluste bei 800 Byte

k800 = 1;

losstemp800 = 0;

while (k800 ~= istpaket800+1)

if(paketcounter800(k800,1)+losstemp800 == istpaketcounter800(k800,1))

paketloss800(k800,1) = istpaketcounter800(k800,1) - (paketcounter800(k800,1)+losstemp800);

end

if(paketcounter800(k800,1)+losstemp800 ~= istpaketcounter800(k800,1))

paketloss800(k800,1) = istpaketcounter800(k800,1) - (paketcounter800(k800,1)+losstemp800);

losstemp800 = losstemp800 + paketloss800(k800,1);

end

k800 = k800 + 1;

end

%Berechnung der Paketverluste bei 600 Byte

k600 = 1;

losstemp600 = 0;

while (k600 ~= istpaket600+1)

if(paketcounter600(k600,1)+losstemp600 == istpaketcounter600(k600,1))

paketloss600(k600,1) = istpaketcounter600(k600,1) - (paketcounter600(k600,1)+losstemp600);

end

if(paketcounter600(k600,1)+losstemp600 ~= istpaketcounter600(k600,1))

paketloss600(k600,1) = istpaketcounter600(k600,1) - (paketcounter600(k600,1)+losstemp600);

losstemp600 = losstemp600 + paketloss600(k600,1);

end

k600 = k600 + 1;

end

%Zusammensetzen aller Daten

%packetloss

counterloss1250 = 1;

while (counterloss1250 ~= istpaket1250)

paketlosstotal(counterloss1250,5) = paketloss1250(counterloss1250,1);

counterloss1250 = counterloss1250 + 1;

end

counterloss1100 = 1;

while (counterloss1100 ~= istpaket1100)

paketlosstotal(counterloss1100,4) = paketloss1100(counterloss1100,1);

counterloss1100 = counterloss1100 + 1;

end

counterloss1000 = 1;

while (counterloss1000 ~= istpaket1000)

paketlosstotal(counterloss1000,3) = paketloss1000(counterloss1000,1);

counterloss1000 = counterloss1000 + 1;

end

counterloss800 = 1;

while (counterloss800 ~= istpaket800)

paketlosstotal(counterloss800,2) = paketloss800(counterloss800,1);

counterloss800 = counterloss800 + 1;

end

counterloss600 = 1;

while (counterloss600 ~= istpaket600)

paketlosstotal(counterloss600,1) = paketloss600(counterloss600,1);

counterloss600 = counterloss600 + 1;

end

%delay

counterdelay1250 = 1;

while (counterdelay1250 ~= istpaket1250)

delaytotal(counterdelay1250,5) = delay1250(counterdelay1250,1);

counterdelay1250 = counterdelay1250 + 1;

end

counterdelay1100 = 1;

while (counterdelay1100 ~= istpaket1100)

delaytotal(counterdelay1100,4) = delay1100(counterdelay1100,1);

counterdelay1100 = counterdelay1100 + 1;

end

counterdelay1000 = 1;

while (counterdelay1000 ~= istpaket1000)

delaytotal(counterdelay1000,3) = delay1000(counterdelay1000,1);

counterdelay1000 = counterdelay1000 + 1;

end

counterdelay800 = 1;

while (counterdelay800 ~= istpaket800)

delaytotal(counterdelay800,2) = delay800(counterdelay800,1);

counterdelay800 = counterdelay800 + 1;

end

counterdelay600 = 1;

while (counterdelay600 ~= istpaket600)

delaytotal(counterdelay600,1) = delay600(counterdelay600,1);

counterdelay600 = counterdelay600 + 1;

end

figure(1);

descloss = {600 800 1000 1100 1250};

boxplot(paketlosstotal,descloss);

title('Packetloss','fontweight','bold','fontsize',10);

xlabel('Declaration in Byte/Packet');

ylabel('Count of lost packets');

figure(2);

descdelay = {600 800 1000 1100 1250};

boxplot(delaytotal,descdelay);

title('Delay','fontweight','bold','fontsize',10);

xlabel('Declaration in Byte/Packet');

ylabel('Delay in milliseconds');

%Datarate 1250Byte in us

transmission1250 = max(startzeit1250(:,1)) - startzeit1250(1,1);

time1250 = 1;

while (time1250 ~= k1250)

transmission1250sec(time1250,1) = (startzeit1250(time1250,1) - startzeit1250(1,1))/1000000;

time1250 = time1250 + 1;

end

totalsenddata1250 = sum(Byte1250(:,5));

datarate1250 = (totalsenddata1250 / transmission1250)\*8;

%Datarate 1100Byte in us

transmission1100 = max(startzeit1100(:,1)) - startzeit1100(1,1);

time1100 = 1;

while (time1100 ~= k1100)

transmission1100sec(time1100,1) = (startzeit1100(time1100,1) - startzeit1100(1,1))/1000000;

time1100 = time1100 + 1;

end

totalsenddata1100 = sum(Byte1100(:,5));

datarate1100 = (totalsenddata1100 / transmission1100)\*8;

%Datarate 1000Byte in us

transmission1000 = max(startzeit1000(:,1)) - startzeit1000(1,1);

time1000 = 1;

while (time1000 ~= k1000)

transmission1000sec(time1000,1) = (startzeit1000(time1000,1) - startzeit1000(1,1))/1000000;

time1000 = time1000 + 1;

end

totalsenddata1000 = sum(Byte1000(:,5));

datarate1000 = (totalsenddata1000 / transmission1000)\*8;

%Datarate 800Byte in us

transmission800 = max(startzeit800(:,1)) - startzeit800(1,1);

time800 = 1;

while (time800 ~= k800)

transmission800sec(time800,1) = (startzeit800(time800,1) - startzeit800(1,1))/1000000;

time800 = time800 + 1;

end

totalsenddata800 = sum(Byte800(:,5));

datarate800 = (totalsenddata800 / transmission800)\*8;

%Datarate 600Byte in us

transmission600 = max(startzeit600(:,1)) - startzeit600(1,1);

time600 = 1;

while (time600 ~= k600)

transmission600sec(time600,1) = (startzeit600(time600,1) - startzeit600(1,1))/1000000;

time600 = time600 + 1;

end

totalsenddata600 = sum(Byte600(:,5));

datarate600 = (totalsenddata600 / transmission600)\*8;

%Berechnung des gesamten paketverlustes bei 1250Byte

Paketlossgesamt1250 = sum(paketloss1250(:,1));

percentloss1250 = (100/sollpaket1250)\*Paketlossgesamt1250;

%Berechnung des gesamten paketverlustes bei 1100byte

Paketlossgesamt1100 = sum(paketloss1100(:,1));

percentloss1100 = (100/sollpaket1100)\*Paketlossgesamt1100;

%Berechnung des gesamten paketverlustes bei 1000byte

Paketlossgesamt1000 = sum(paketloss1000(:,1));

percentloss1000 = (100/sollpaket1000)\*Paketlossgesamt1000;

%Berechnung des gesamten paketverlustes bei 800byte

Paketlossgesamt800 = sum(paketloss800(:,1));

percentloss800 = (100/sollpaket800)\*Paketlossgesamt800;

%Berechnung des gesamten paketverlustes bei 600byte

Paketlossgesamt600 = sum(paketloss600(:,1));

percentloss600 = (100/sollpaket600)\*Paketlossgesamt600;

figure(3);

[AX,H1,H2] = plotyy(transmission600sec,delay600,transmission600sec,paketloss600,'plot','bar');

axis([AX(1) AX(2)],[2 3 0 180]);

set([AX(1) AX(2)],'YTick', 0:20:200);

title('Delay and Packetloss with 600Byte/Packet','fontweight','bold','fontsize',10);

xlabel('transmission time in seconds');

set(H2,'LineStyle','-');

set(H1,'Color','red');

set(AX(1),'ycolor','red');

set(AX(2),'ycolor','blue');

ylabel(AX(1),'Delay/packet in milliseconds') % left y-axis

ylabel(AX(2),'Counted packetloss per intervall') % right y-axis

figure(4);

[AX,H1,H2] = plotyy(transmission800sec,delay800,transmission800sec,paketloss800,'plot','bar');

axis([AX(1) AX(2)],[2 3 0 180]);

set([AX(1) AX(2)],'YTick', 0:20:200);

title('Delay and Packetloss with 800Byte/Packet','fontweight','bold','fontsize',10);

xlabel('transmission time in seconds');

set(H2,'LineStyle','-');

set(H1,'Color','red');

set(AX(1),'ycolor','red');

set(AX(2),'ycolor','blue');

ylabel(AX(1),'Delay/packet in milliseconds') % left y-axis

ylabel(AX(2),'Counted packetloss per intervall') % right y-axis

figure(5);

[AX,H1,H2] = plotyy(transmission1000sec,delay1000,transmission1000sec,paketloss1000,'plot','bar');

axis([AX(1) AX(2)],[2 3 0 180]);

set([AX(1) AX(2)],'YTick', 0:20:200);

title('Delay and Packetloss with 1000Byte/Packet','fontweight','bold','fontsize',10);

xlabel('transmission time in seconds');

set(H2,'LineStyle','-');

set(H1,'Color','red');

set(AX(1),'ycolor','red');

set(AX(2),'ycolor','blue');

ylabel(AX(1),'Delay/packet in milliseconds') % left y-axis

ylabel(AX(2),'Counted packetloss per intervall') % right y-axis

figure(6);

[AX,H1,H2] = plotyy(transmission1100sec,delay1100,transmission1100sec,paketloss1100,'plot','bar');

axis([AX(1) AX(2)],[2 3 0 180]);

set([AX(1) AX(2)],'YTick', 0:20:200);

title('Delay and Packetloss with 1100Byte/Packet','fontweight','bold','fontsize',10);

xlabel('transmission time in seconds');

set(H2,'LineStyle','-');

set(H1,'Color','red');

set(AX(1),'ycolor','red');

set(AX(2),'ycolor','blue');

ylabel(AX(1),'Delay/packet in milliseconds') % left y-axis

ylabel(AX(2),'Counted packetloss per intervall') % right y-axis

figure(7);

[AX,H1,H2] = plotyy(transmission1250sec,delay1250,transmission1250sec,paketloss1250,'plot','bar');

axis([AX(1) AX(2)],[2 3 0 180]);

set([AX(1) AX(2)],'YTick', 0:20:200);

title('Delay and Packetloss with 1250Byte/Packet','fontweight','bold','fontsize',10);

xlabel('transmission time in seconds');

set(H2,'LineStyle','-');

set(H1,'Color','red');

set(AX(1),'ycolor','red');

set(AX(2),'ycolor','blue');

ylabel(AX(1),'Delay/packet in milliseconds') % left y-axis

ylabel(AX(2),'Counted packetloss per intervall') % right y-axis

fprintf('The datarate with 600Bytes is: %.2f MBit/s\n', datarate600);

fprintf('The datarate with 800Bytes is: %.2f MBit/s\n', datarate800);

fprintf('The datarate with 1000Bytes is: %.2f MBit/s\n', datarate1000);

fprintf('The datarate with 1100Bytes is: %.2f MBit/s\n', datarate1100);

fprintf('The datarate with 1250Bytes is: %.2f MBit/s\n', datarate1250);

fprintf('The packetloss with 600Bytes is by: %d Packets\n', Paketlossgesamt600);

fprintf('The packetloss with 800Bytes is by: %d Packets\n', Paketlossgesamt800);

fprintf('The packetloss with 1000Bytes is by: %d Packets\n', Paketlossgesamt1000);

fprintf('The packetloss with 1100Bytes is by: %d Packets\n', Paketlossgesamt1100);

fprintf('The packetloss with 1250Bytes is by: %d Packets\n', Paketlossgesamt1250);

Script 10: MATLAB script for evaluation