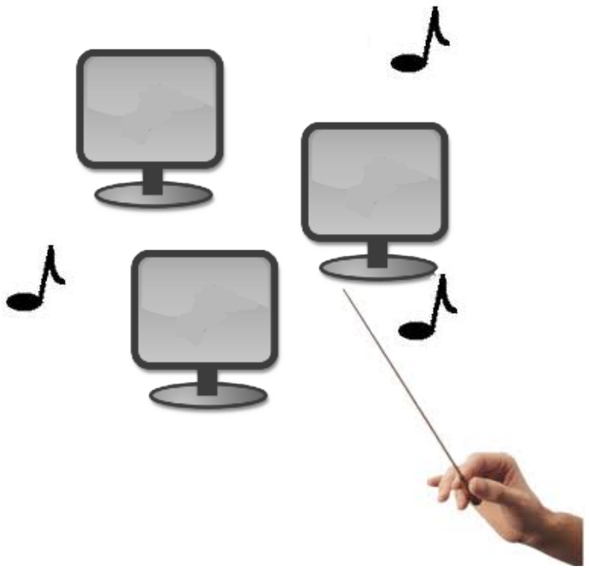
CHALMERS



Single point of Control for a Distributed System
Centralized Control of a Real-Time Ethernet Measurement System

Master of Science Thesis in Networks and Distributed Systems

FERNANDO FERRI VIDAL

Chalmers University of Technology
Department of Computer Science and Engineering
Göteborg, Sweden, October 2011

Single Point of Control for a Distributed System
Centralized Control for a Real-Time Ethernet Measurement System

FERNANDO FERRI VIDAL

Examiner: JAN JONSSON
Supervisior: JONAS LEXT
Co-worker: FATEMEH AYAT

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden October 2011.

Abstract

TCN Analyzer is a design and analysis tool that allows the user to create a model of a network to be analyzed. It provides a prediction of the maximum forwarding time in the network for a concrete flow and some other properties such as the guarantee of non-packet drops. The development of TCN Analyzer is still in a stage where it is needed to verify that the mathematical formulas used in the schedulability analysis make correct predictions about the forwarding times of each frame.

In order to verify the predictions, at the start up of this project the company had a manual measurement procedure formed by several manual tasks. It needs to be automated and improved in this project.

Finally, at the end of this project, a centralized entity that automate the measurement system is developed successfully, allowing the user to obtain the timing results just by executing the developed measurement tool.

Keywords: Distributed System, Ethernet Measurement System, Time Critical applications, Real-Time scheduling, TCN Analyzer, Network design.

Acknowledgements

This has been a very intensive project and I would like to thank all the people that directly or indirectly participated in it. It has been carried out under supervision of Jan Jonsson and Jonas Lext, whose guidance and encouragement have been crucial to finish this project successfully.

To start I want to give my best gratitude to Fatemeh Ayat, my friend and co-worker in this project, without who this would not have been possible and who contribute to make this project an enjoyable experience. As well I want to thank Mohammad Ibrahim for his help and advices at the time of programming the boards.

This has been the first time I worked in a project for a company and I would like to thank Lars Bröhne, Jonas Lext and Thomas Lundqvist for their support and for giving me the chance to take part in it. They were very helpful and understanding, which made working with this project a lot more enjoyable.

Finally, and not less important, I want to thank my friends and family strongly for the support and affection given to me during my thesis period and these two years of my Master studies I have spent abroad.

Table of Contents

List of Figures

0. Method

This section pretends to describe the method in which the tasks and requirements were established for the development of this project.

Due that this is a project for a company, which aims to solve a specific need, a meeting with the managers of the company took place in order to find out the requirements and define which tasks to perform.

This is reflected in the *Introduction* section. It pretends to describe the company's product, TCN Analyzer, in the subsection *TCN Analyzer tool,* and define the problem that this project has to face in the subsection *Purpose of the thesis work*.

First of all, it was decided to do a pre-study of the system, described in *section 2.* The aim of this was to obtain a clear insight about the initial system in order to define properly the requirements of this project.

After the pre-study phase all the manual steps of the manual measurement process were clearly defined. This was really important in order to decide which tasks to do for performing the automation of the measurement system, the aim of this thesis project.

Finally, together with the managers of the company it was decided the road map of this project. It is described in *section 3, Design and Implementation,* and it is structured in two parts:

A) Software Development: develop a coordinator program, which will interact with the different parts of the system and provide the results.

For this section it was defined a set of specific tasks to be done in order to overcome some issues:

- Avoid moving the capture file between computers. (3.1)
- Overcome the need of format conversions. (3.3)
- Avoid the need of interact with graphical interfaces, GUIs. (3.2)

After the commitment of these tasks, the main limitations for the automation of the system would be solved. This will make possible to develop the coordinator program, which will control all the tasks needed for performing and analysing a measurement.

B) Provide a suitable system architecture: define and develop the appropriate network architecture, separating the control part from the measurement part.

1. Introduction

This section focuses on two different protocols used for transmitting into a communication medium from the point of view of time critical systems, CAN and Ethernet (1.1). Finally it introduces TCN Analyzer tool, related to the design and development of Ethernet networks (1.2) and it concludes explaining the purpose of the thesis work (1.3).

1.1 Ethernet vs. CAN in time critical systems

Hosts need a protocol to define how to transmit into a communication medium, being a bus, a coaxial cable used for Ethernet or another technology. These protocols handle how and when to initiate the transmission in shared medium, defining the properties in which data is transmitted and which measures are taken in order to provide reliable services such error detection or fault tolerance.

The **Controller Area Network (CAN)** is a vehicle bus communication protocol created originally for Robert Bosch in 1983 for the transmission of data in critical applications in which was required fault tolerance, allowing microcontrollers and devices to communicate within a vehicle without the interaction of any computer. It supports distributed real time control with a very high level or security [17].

These are some of the most important CAN propieties:

- Priorization of messages
- Guarantee of latency times
- Multicast reception with time synchronization
- Error detection and error signalling
- Automatic retransmission of the messages that are corrupted
- Fault tolerance management and confinement. Defect nodes are switched off.

CAN is used mainly in automation and as well is being used in other sectors as trains or airplanes where fault tolerance and safety constrains are a major issue. In order to achieve these powerful measures for error detection, signalling and detection of faults are implemented in every CAN node.
It ensures that the message with higher priority is always sent when several units try to transmit at the same time. This is called priority based bus arbitration, where the flow with higher priority is allowed to transmit. Other units, which transmit lower priority flows, detect it. Therefore higher priority flows are transmitted first and the others will wait.

CAN is used in embedded systems and automotion systems [8], being possible find it in:

- Trucks control systems
- Train systems
- Marine systems
- Industrial automotion
- Elevators
- Medical equipment

CAN is a protocol used in time critical environments where integrity, fault tolerance and latency constraints are required. In case of not fulfil these requirements safety could be compromised and it could be a risk for human lives.

On the other hand CAN protocol has some limitations and propieties:

- Low bit rate, where maximum velocity is 1 Mbit/sec.
- Throughput depends on bus length, achieving the maximum velocity up to 30 m.
- It has a non-destructive arbitration when more than one node pretends to transmit. The node with highest priority still will be able to transmit their message.

CAN limitations, specially the low bit rate or velocity in which data can be transmitted and the relatively short length of the buses used for transmission brings up the necessity of replacing this protocol which is the one used nowadays mainly in the automotive industry for a better and more efficient protocol.

On the other hand, **Ethernet** is a networking protocol for Local Area Networks (LAN), introduced in 1980 and standardized in IEEE 802.3 [12].

It provides better propieties such a higher data rate than CAN, increased from 10 Mbits/ sec originally to 1000 Mbits/sec.

Nevertheless Ethernet does not provide by itself fault tolerance or integrity services except allowing discarding corrupted frames if any is detected. It just provides the service doing the best effort for it. In case of some data is corrupted or does not reach the destination, other higher-level protocols such is TCP should be used for allowing a reliably transmission system.

In this case the length of the medium is not a problem, being able to transmit up to very long distances.

The main propieties to highlight about Ethernet are:

- Maximum bit rate 10/100/1000 Mb/sec, higher bit rates can be achieve with new technological improvements such as Gigabit Ethernet [13].
- Source and destination for a message, not only broadcast.
- Arbitration is destructive; if more than one node transmits at the same time no useful data can be recovered.

In industry there is interest in find out the actual capacity, reliability and usability of Ethernet Networks in the industrial control area. The article authored by Jeff Nowling , Industrial Automotion Engineering Consultant [18], focuses on this questions, how fast is possible to communicate via Ethernet, the effects on reliability and determinism. This article highlight that a good network throughput depends on the network design, bit rate, speed in which data is transmitted and remains in the network and other devices used in the network.

Nevertheless it concludes that a properly planned and installed Ethernet network should be capable of providing fast, reliable and deterministic transmission of data in industrial control applications.

There are many reasons for adopting Ethernet standard in these applications [18]:

- Ethernet is nowadays a technology very widespread, operating in reasonable good costs.
- It has a good performance in terms of bandwidth and bit rate, transmitting big amounts of data at high speed.
- Interoperability and convergence. It allows multiple services on a single cable.
- Standards well widespread and adopted. It would bring the advantage in interacting with another systems or networks, due that most of them operate in Ethernet no conversion system would be needed.

On the other hand, industrial Ethernet would be lacking certain propieties that are significant for industrial applications.

These are:

- Guarantee that frames are transmitted within a certain time bound (latency).
- Fault tolerance and redundancy mechanisms that act when a unit fails.
- Determinism always is predicted the same results certain circumstances, there are not undefined status.

New technology is available nowadays that tries to solve these issues. TCN-Analyzer overcomes these restrictions. TCN tool provides determinism to the standard Ethernet with latency, jitter and packet-drop guarantees.

1.2 TCN-Analyzer tool.

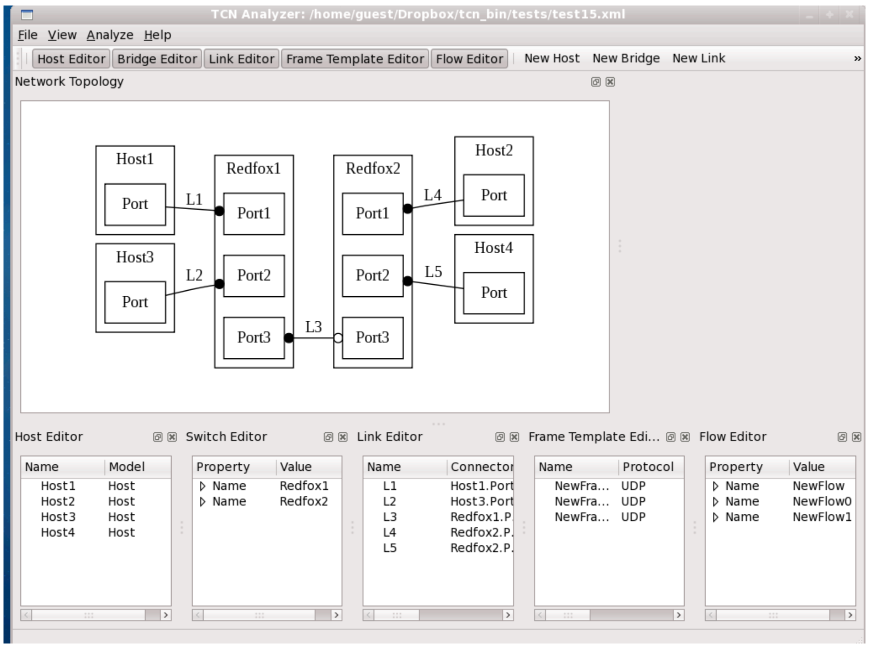
TCN-Analyzer tool is a tool developed by Time Critical Networks AB. TCN's prototype design- and analysis tool, TCN Analyzer, takes advantage of schedulability analysis theory for store-and-forward, multi-hop Ethernet networks [1].

Instead of performing measurements on a network comprised of many switches and populated with many different flows, TCN advocates the following approach to network design- and analysis:

1. Perform measurements on different types of components in the network in isolation.
2. From these measurements, create timing models that describe how the component behaves under different circumstance.
3. Incorporate the timing models into TCN Analyzer were one can construct a complete network using virtual switches and flows and then compute upper bounds on the forwarding time of individual frames, for example, an UDP frame in a data flow sent from one host-computer to another.

It allows the user to define a network adding virtual links, flows, switches and hosts where it is possible to define their timing propieties such as link speed or size of the flow packets.

The GUI of this program is shown in *Figure 1*, where are defined four hosts and two Redfox Switches all together with the links that connect the network and the flow definition.



Name	Model
Host1	Host
Host2	Host
Host3	Host
Host4	Host

Property	Value
▷ Name	Redfox1
▷ Name	Redfox2

Name	Connector
L1	Host1.Port
L2	Host3.Port
L3	Redfox1.P.
L4	Redfox2.P.
L5	Redfox2.P.

Name	Protocol
NewFra...	UDP
NewFra...	UDP
NewFra...	UDP

Property	Value
▷ Name	NewFlow
▷ Name	NewFlow0
▷ Name	NewFlow1

Figure 1: TCN-Analyzer GUI

Once the virtual network is created and their propieties are set up, the program will be ready for performing the timing analysis. This timing analysis, which is the aim of this tool, provides an estimation of the maximum time needed for a packet or a flow to reach their destination. **TCN-Analyzer calculates an upper bound of the transmission delay** or the maximum forwarding time. The tool makes a prevision of this upper bound time and guarantees that any packet of the flow do not exceeds this estimated time. In case that the system does not guarantees that all the packets are schedulable, the designer can make changes in the system modifying the links speed, priorities and the topology until the schedulability is verified by TCN-Analyzer.

This upper bound time is what users are interested to know in order to ensure and verify that their time-critical applications in the network meet the latency requirements. TCN-Analyzer makes their prediction for Ethernet networks.

Nowadays most of time-critical networks, like the control networks in a car, use CAN as a transmission protocol. Nevertheless, CAN has certain limitations as described.
It is considered in industry that most of the time critical applications using CAN will be replaced by Ethernet networks [9][10].

TCN-Analyzer pretends to be a verification tool for this promising technology, taking advantage about the lack of applications that work in this field and the growing interest about using Ethernet networks in time-critical systems such as in automation.

Facts which confirm the growing interest that industry put into this technology are the project performed by CPAC Systems, company dependant of Volvo, together with Time Critical Networks AB aiming to check the feasibility or replacing CAN with standard Ethernet [11] and the research groups of BMW that test the use of Ethernet in their systems.

BMW engineers tested the use of IP applications, using Ethernet protocol- to network automotive controllers in the engine control unit, founding that IP-Ethernet could suit well the real-time requirements even for safety-critical applications. It is considered that modern cars incorporate over 70 embedded computers networked by a host of different systems. Automotive industry turns their attention to Ethernet as a common backbone for the car network, offering an important simplification of the automotive networking jungle [9].

In conclusion, it is predicable that Ethernet will be the transmission protocol used in a not so far future in time critical applications, which use CAN nowadays. Most of the tools currently available for verifying the timing constraints and correctness of the networks used in time-critical applications are developed for CAN networks.

Therefore TCN-Analyzer, which is a tool for designing and verifying the timing correctness in Ethernet networks has chances to be a successful tool for the usage of Ethernet networks in this field as described, where there is a lack of products that deal with Ethernet in time-critical network analysis.

1.3 Purpose of Thesis Work

The aim of this thesis work is not related directly with the main tool previously described, TCN- Analyzer. However this project focuses on designing and implementing a Centralized control for a real-time Ethernet Measurement System, which will provide the tool (software) and the infrastructure (networks) needed for performing the verification of the timing predictions provided by TCN-Analyzer. It pretends to be an automatized measurement system, where the results could be obtained with the minimum user interaction.

A measurement system was available at the start up of this project, nevertheless this measurement system needed of several tasks to be done sequentially in order to perform a single measurement. These tasks were manual, including interaction with different computers, where the data obtained in a computer needed to be sent manually to another computer in order to perform the next task, in the same single measurement. Furthermore, these tasks included interaction with GUIs, graphical interfaces with mouse usage and help of some externals programs for converting some files to another format.

This procedure made the measurement not user friendly, a difficult process and not really efficient to obtain measurements, where it could take several minutes to perform the set of manual tasks for a single measurement.

Next section gives an overview of the TCN's initial measurement system and its individual components.

2. Pre-study phase

The aim of this section is to provide a big picture about the initial measurement system, which are the different units and how they interact between them defining the interfaces and describing the different tasks that needed to be done for performing a measurement.

2.1 System Overview

The development of this project takes place in TCN's office, where a private lab network, also called measurement network separated from the office network have been created exclusively for the project development.

Next figure, *Figure 2* shows the network topology:

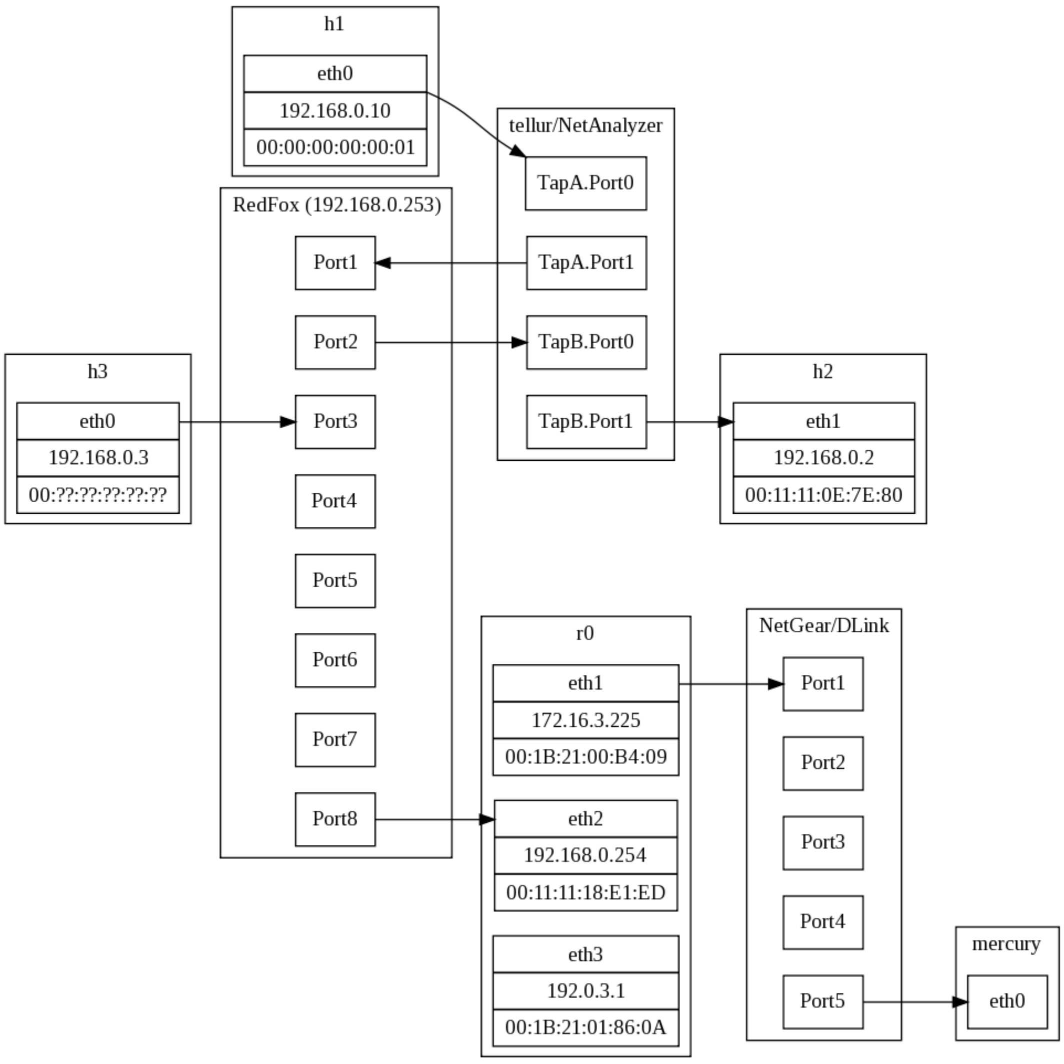


Figure 2: network topology

The router, *R0*, is the gate that separates networks, the lab network with interface *eth2* and the office network with interface *eth1*. To *eth1* is connected the router *DLink* that provides connection to all hosts in the office network.

On the other hand, *eth2* is connected the Redfox switch, to which are connected the different hosts (h1, h2, h3) used for generating and collecting the packets and the real-time *NetAnalyzer card*, which measures the packets forwarding time in the previous switch.

This is the basic architecture of the network created for the project start up and it has been improved and redesigned according to the project development.

2.1.1 Traffic flow generation

In order to generate the traffic flow in hosts *h1* and *h3* located at the lab network, an external computer located at the office network (*mercury)* is used.

Through *SSH* in mercury we access to the hosts that generate the traffic flow. The hosts in the lab network, called as well measurement network, use the open source project Click [2], a configurable software router developed and managed by a research group at a university in USA. The TCN Click engine has been slightly improved with a custom C++ class that simulates the IP-stack of a normal computer. When a Click-script calls this class it can generate idealized bursts of maximum sized Ethernet packets corresponding to what large UDP-frames would have been fragmented into by the IP-stack of a normal computer. These Click-machines are, unfortunately, currently not good at producing flows of minimum sized Ethernet frames without substantial jitter.

The destination of the flows is the host *h2*, which is running the Click project and a script that counts the number of packets received from *h1* and *h3*. This host provides the possibility of accessing to it through a own terminal (keyboard and screen) located in the lab network, therefore it is not necessary access to it through SSH from the office network.

Flow description:
Host *h1* sends packets that are captured by the NetAnalyzer card, adding a time stamp and sending them later to the switch. The forwarding time at the switch is calculated when the packets are sent back to the NetAnalyzer card using the previous time stamp.
Next, the packets are stored in a hard disk, where different format conversions have to be done before the data is sent to the Stream-Analyzer for calculating the forwarding time. Furthermore, all packets are sent to the host *h2,* where they are classified and counted depending of which host they come from.

At the same time, it is possible to generate another data flow in the host *h3.* This will introduce another data flow competing for forwarding time in the switch making the packets being stored into a queue before being forwarded. *Figure 3* shows the packet flow in the system.

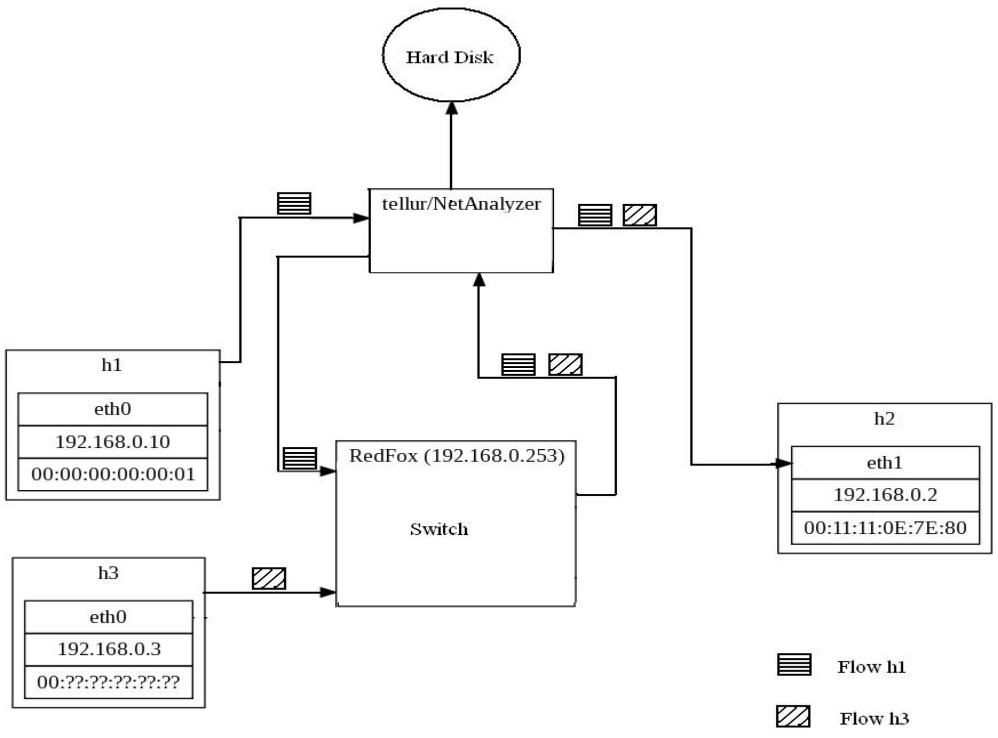


Figure 3: Traffic flow

In Figure 3 is shown how are connected the hosts in order to measure the packet flow providing from *h1*. In this case *h3* has been introduced for generating traffic that could interfere in the switch and increase the switch forwarding time.

The NetAnalyzer card has four interfaces as is shown in its GUI provided in the section 2.2.1. This can be observed here looking at the four arrows that have as a source or destination the NetAnalyzer card in *Tellur (host windows computer)*.

The first interface belongs to the flow that we pretend to measure, in this case *h1*. So in this first step the packets that came from *h1* are stamped with a time mark by the NetAnalyzer card and send back to the switch through the second interface. Next interface, the third one sends the packets coming from the switch back to the card, where the messages previously tagged with a time stamp are detected and its forwarding time is calculated.
Finally the messages are sent to the sink or destination through the forth interface.

Once that all the different components of the system have been introduced, the next step is to describe the procedure of obtaining the forwarding times, which it includes: generating the data flow, how it is captured at the destination and how it is captured and analysed by Stream-Analyzer after several format conversions by the different units, describing the different inputs and outputs in this procedure. This is by now done manually, which automation is the aim of this master thesis.

2.1.2 Manual measurement procedure

The setup configuration for the network, including issues such as routing table configuration and system setups are included in the Appendix A. Notice that this settings need to be done manually each time the system is started in order to performing measurements. The aim is to avoid these manual procedures at the end of this project and provide a centralized measurement control system that performs all the tasks without needing the user interaction.

A) Boot lab network

Start the machines *r0*, which act as a router, *h1*, and *h3*, which act as a traffic generators and *h2*, which acts as a receiver of the data-flow.

It is important before starting the capture of packets make a Ping call to all different hosts in the system, make a Ping from *h1* and *h3* to *h2* and vice versa. This is done for updating the switch table and avoid that packets are broadcasted if the switch does not have stored from which port number is reachable the desired host. This is done in every machine, what implies a considerable time waste.

B) Start the packet capture using the Card- Analyzer GUI.

The traffic delivered by *h1* goes directly to host *Tellur*, containing the Real-Time network card, which captures the packets, add a time stamp before forwarding them to the switch.
This card only provides a graphical interface (GUI) where it has to be pressed the button START for initiating the capture of packets.

Before generating the packet flow in h1 and h3 is necessary to press the button START from the GUI, then the NetAnalyzer driver routine will start saving the capture measurement in the file *C:\ Default.hea*.

C) Collect packets at sink host

Host *h2* is the sink node, which collects all packets sent from *h1* and *h3*. Host *h2* is operated directly via the connected keyboard and monitor.

Therefore this needs a manual interaction with the computer *h2* in order to execute the commands that will enable the sink click program which will receive and count the packets.
This steps include access to the Click folder, where the next command have to be executed in order to start collecting the packets (run the Click program that capture the packets): *click-install /root/click/sink_cec.click.*

D) Generating the traffic using Click

Once that all hosts are accessible after setting up the routing tables and checking that all hosts are accessible using PING, *Mercury*, the host placed in the office network is used for accessing through *SSH* to the hosts *h1* and *h3* (traffic generators) placed in the lab network.

Through *Mercury,* computer placed in the office network we log on to *h1* placed in the measurement network with the command SSH root@192.167.0.1 (*h1* IP address) using the corresponding password.

Through *SSH* we access to the Click folder, where we should execute the script *FastUDPSource.click*. This script contains the setup for sending certain number of packets, data rate and packet length. This is done by the command: *click-install FastUDPSource.click*. This command will start the traffic generation from *h1*.

At the same it is possible to create another data flow from the host *h3* in the same way, login on from *Mercury* to *h3* using *SSH* with the command root@192.167.0.3 with the corresponding password. The aim of this data flow is to cause collisions between the traffic from *h1* and *h3*, where some packets should be stored into the switch queue increasing the forwarding time. Later on in the analysis will be possible to analyse in which moment collisions take place and what is the forwarding time for these packets.

E) Stop the packet capture using the Card- Analyzer GUI.

Once it has been decided that enough traffic have been captured in order to perform the analysis, it needs to be pressed the STOP button at the NetAnalyzer GUI. It will stop capturing packets and the capture will be available in a binary file with extension . *hea*.

F) Obtain number of packets received in sink

Next, at the end of the capture and after being pressed the stop button, we will be able to obtain how many packets have been received in *h2* executing the script that counts the received packets from *h1* and *h3*:
cat /click/count_from_h1/count /click/count_from_h3/count

This command will print in the first line the number of received packets from *h1* and in the second line the number of received packets from *h3*.

In order to turn off Click after using it the next command can be executed: *click-uninstall.*

G) First Format Conversion.

NetAnalyzer's driver stores the capture file in a propietary format, a binary file with *.hea* extension. Next step is to convert the format to *.pcap [14]* using the option Convert in the menu of the application provided by Hilscher *(Figure 4)*. The same application (GUI) allows to start, stop a capture and convert the resulting file to *.pcap*, a format that is readable by *Wireshark* [15]. This new file in *.pcap* format can be readed with *Wireshark*, which shows all the captured packets during the measurement.

H) Second Format Conversion.

To analyse the measurement with Stream-Analyzer the previous file in *.pcap* format must be translated to *.csv* format. This is done by the option *Export file* at the graphical interface of Wireshark *(Figure 5)*. This *.csv* format is more readable from a human point of view and it is the format used by Stream-Analyzer, program developed in C++ by Jonas Lext in order to process the capture and obtain the timing results such as the maximum forwarding time that we are interested.

I) Move capture files to a Linux machine that runs Stream-Analyzer

Once this last format translation is done, the resulting file containing the measurement have to be moved using some manual method such as mail or Dropbox to another Linux machine that runs the Stream-Analyzer using the captured data in *.csv* format for performing the forwarding time analysis.

One of the aims of this project is to avoid such format transformation and file movement from different hosts in order to perform the system automation.

J) Analysing the measurement with Stream Analyzer

After obtaining the measured data from the Measurement card, it is time to analyse the data and find out about the desired features of the flow, like the forwarding times in the switch. This is done by StreamAnalyzer program, which is a C++ program, which runs only in Linux machines. So after the capture file is obtained, several format conversions are done, obtaining finally the *.csv* file, which is readable by StreamAnalyzer. Now we can use StreamAnalyzer in order to classify the packets and separate them into different applications flows depending on protocol, source and destination address etc. Furthermore, it can compute the forwarding times of the packets that passed the two taps of the measurement card and can produce different types of plots.

In this way, the StreamAnalyzer program use two files when it runs. The command to run the StreamAnalyzer is:

```
$streamanalyzer -ub 100 -fw 70 -sas  <scriptfile>.sas   <measuredfile>.csv
```

The *.scv* file, contains the captured flow and the script file *.sas,* contains the instructions of the desired analysis. This script tells the StreamAnalyzer about what we are looking for and which graphs are desired. Basically, we are looking for these features:

- Forwarding time for the Ethernet packets from TapA (interface 1 and 2 of the NetAnalyzer card) to TapB (interface 3 and 4).
- Serialization of the packets from *h1* and *h3* at the maximum forwarding time in TapB.
- Maximum inter-arrival time. The difference in inter-arrival time causes the jitter delay. So we are looking for the flow behaviour when this inter-arrival time increases.

A current bottleneck is that the file produced by the measurement card must be converted manually twice, first from the propietary Hilscher binary file format into the *.pcap* binary format. Then again from *.pcap* to *.csv*, which is a simple text-based format. Finally, StreamAnalyzer reads the text file after being moved manually to a Linux machine in which StreamAnalyzer can be executed.

Hilscher documentation should explain the structure of the Hilscher binary file format, so StreamAnalyzer should be able to read this format directly in an advanced stage of the project.

2.2 Different Interfaces

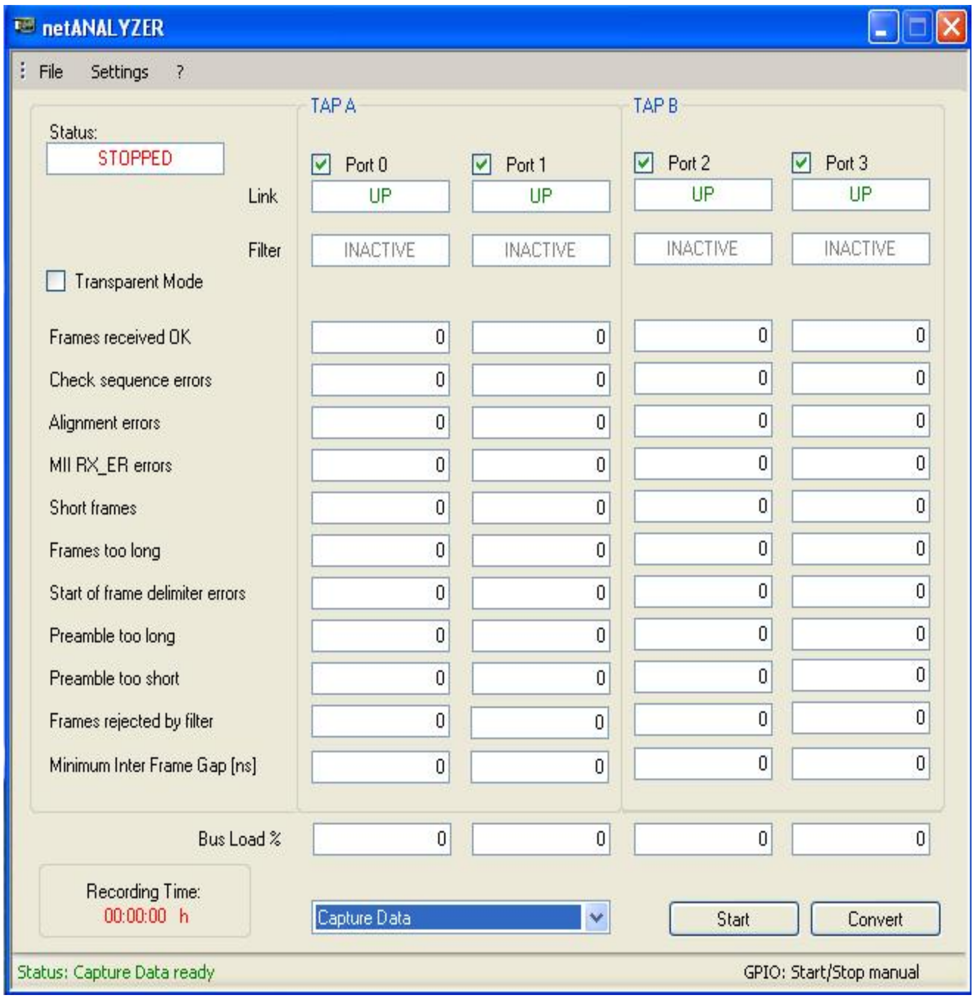
2.2.1 Real-Time Ethernet Measurement card Interface

To capture streams of Ethernet packets, we use a NXANL 50-RE measurement card from Hilscher [3]. The card passively analyses the data in a network communication link and captures the incoming Ethernet frames. Before sending each captured frame to a file on the hard drive, the card inserts a time stamp with nano second resolution into the packet header. By tapping the Ethernet packets just before they enter a switch, and again as soon as they leave the switch, it is possible to compute the forwarding time of the packet through the switch.

The data capturing process must be configured and started via the NetAnalyzer software that comes with the card and which is installed on a Windows XP PC. However, from Hilscher we have acquired documentation that briefly explains what functions are available in the card driver that is residing on the machine. This documentation should be enough to be able to call the driver functions from a custom developed application. Thus, it should be possible to write an application that allows remote starting and stopping of a capturing session. Currently, analysis of the captured stream is done off-line using the application StreamAnalyzer discussed in the next section.

The interface, which is used for the measurement card, is called NetAnalyzer software. There exists a graphical interface shown in *Figure 4*. This is used to start and stop capturing packets and also configuring the filters. This graphical interface is compatible with Windows XP, Vista and 7 operating systems, so we used Windows XP PC for the Measurement card interface. The captured data is saved on the machine hard disk in *.hea* format, but the interface has the option to convert it to other formats like *.pcap* to be readable with other programs like *Wireshark*.

GUI used for the user in order to generate a capture (Start button) , to stop capturing packets (Stop button) and converting the format (Convert button) :



Status: STOPPED

☐ Transparent Mode

	TAP A		TAP B	
	✓ Port 0	✓ Port 1	✓ Port 2	✓ Port 3
Link	UP	UP	UP	UP
Filter	INACTIVE	INACTIVE	INACTIVE	INACTIVE
Frames received OK	0	0	0	0
Check sequence errors	0	0	0	0
Alignment errors	0	0	0	0
MII RX_ER errors	0	0	0	0
Short frames	0	0	0	0
Frames too long	0	0	0	0
Start of frame delimiter errors	0	0	0	0
Preamble too long	0	0	0	0
Preamble too short	0	0	0	0
Frames rejected by filter	0	0	0	0
Minimum Inter Frame Gap [ns]	0	0	0	0
Bus Load %	0	0	0	0

Recording Time: 00:00:00 h

Capture Data

Status: Capture Data ready

GPIO: Start/Stop manual

Figure 4: Hilscher 's Measurement Card Interface (GUI)

The filter settings can be used to accept or reject packets from special source and destination MAC addresses. The settings can be saved and used later as well.

There exist two analysing options, one is for capturing packets and save them on the hard disk, so they can be used later to analyse the time and other features. The second option is the timing analysis.

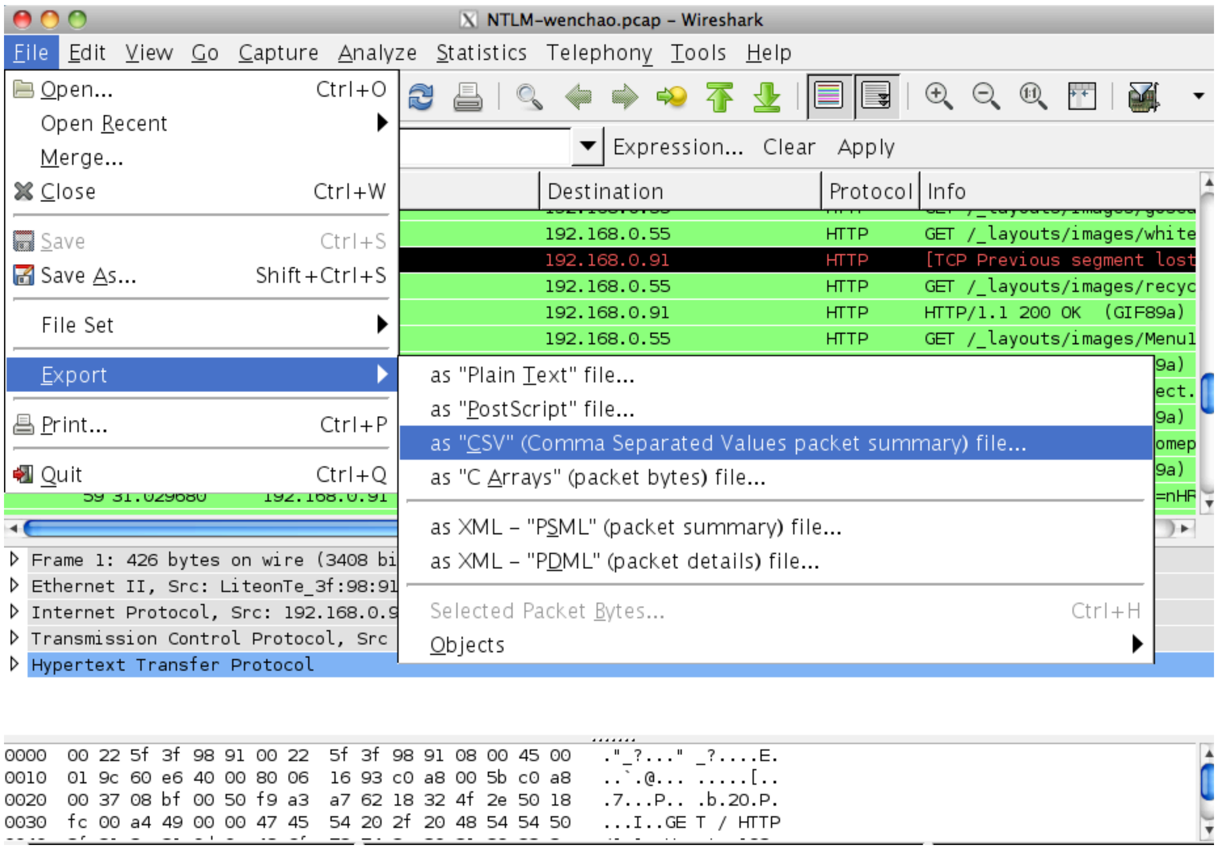
In this project as mentioned in section 2.1, we are going to estimate the Forwarding time in the Switch, so we use the *Capture Data* for capturing every packet, being stored later into a file which we convert to *.pcap* format to use it later in *Wireshark* and get the timestamps of the packets before and after the switch.

2.2.2 Wireshark Interface (second file conversion process)

Next figure shows the graphical interface of Wireshark, program that is needed in order to perform the second file conversion process.

The first conversion process is done through netAnalyzer GUI shown in *Figure 4* as described. It converts the capture file from *.hea* format to *.pcap* format, understandable by Wireshark.

Finally in this second conversion process using Wireshark, the *.pcap* file is converted to *.csv* format as it is shown in *Figure 5*, which is readable by StreamAnalyzer, concluding the file conversion procedures.

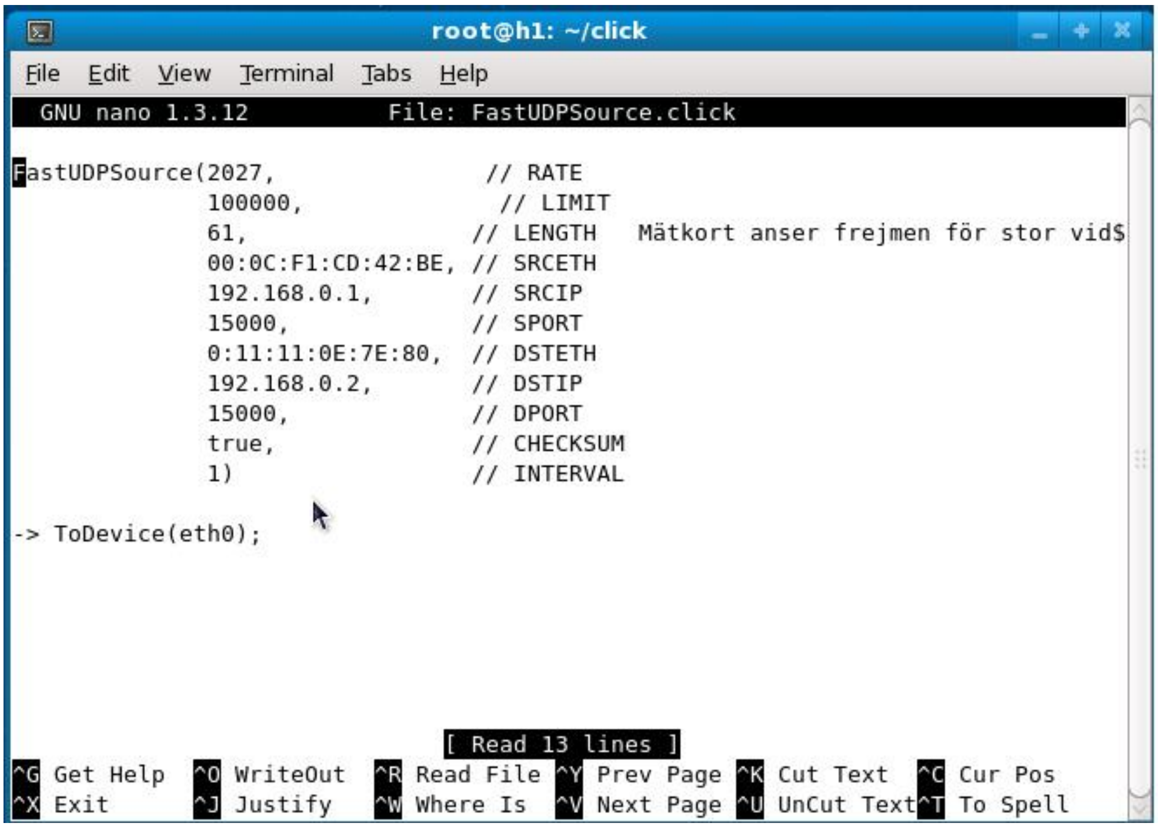

NTLM-wenchao.pcap – Wireshark
File Edit View Go Capture Analyze Statistics Telephony Tools Help
Open... Ctrl+O
Open Recent
Merge...
Close Ctrl+W
Save Ctrl+S
Save As... Shift+Ctrl+S
File Set
Export
Print... Ctrl+P
Quit Ctrl+Q
Expression... Clear Apply
as "Plain Text" file...
as "PostScript" file...
as "CSV" (Comma Separated Values packet summary) file...
as "C Arrays" (packet bytes) file...
as XML – "PSML" (packet summary) file...
as XML – "PDML" (packet details) file...
Selected Packet Bytes... Ctrl+H
Objects

```
Destination      Protocol  Info
192.168.0.55     HTTP      GET /_layouts/images/white
192.168.0.91     HTTP      [TCP Previous segment lost
192.168.0.55     HTTP      GET /_layouts/images/recyc
192.168.0.91     HTTP      HTTP/1.1 200 OK  (GIF89a)
192.168.0.55     HTTP      GET /_layouts/images/Menu1
59 31.029680   192.168.0.91

Frame 1: 426 bytes on wire (3408 bi
Ethernet II, Src: LiteonTe_3f:98:91
Internet Protocol, Src: 192.168.0.9
Transmission Control Protocol, Src
Hypertext Transfer Protocol

0000  00 22 5f 3f 98 91 00 22  5f 3f 98 91 08 00 45 00   ."_?..." _?....E.
0010  01 9c 60 e6 40 00 80 06  16 93 c0 a8 00 5b c0 a8   ..`.@... .....[..
0020  00 37 08 bf 00 50 f9 a3  a7 62 18 32 4f 2e 50 18   .7...P.. .b.2O.P.
0030  fc 00 a4 49 00 00 47 45  54 20 2f 20 48 54 54 50   ...I..GE T / HTTP
```

Figure 5: Wireshark format conversion

2.2.3 Click Interface

The Click software is used to generate the packet flow. In this way, there is no graphical interface, and we use the Unix terminal and Click-install script as an interface to generate the desired flow. For instance, there is a function called FastUDPSource as shown in *Figure 6* which can set the flow parameters like rate, number of packets, length, address and port of source and destination, etc. By running the click-install script on this function, the flow will be generated on the network.



```
  GNU nano 1.3.12           File: FastUDPSource.click

FastUDPSource(2027,              // RATE
              100000,             // LIMIT
              61,               // LENGTH   Mätkort anser frejmen för stor vid$
              00:0C:F1:CD:42:BE, // SRCETH
              192.168.0.1,      // SRCIP
              15000,            // SPORT
              0:11:11:0E:7E:80,  // DSTETH
              192.168.0.2,      // DSTIP
              15000,            // DPORT
              true,             // CHECKSUM
              1)                // INTERVAL

-> ToDevice(eth0);




                              [ Read 13 lines ]
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

Figure 6: Click interface, program code executed at the terminal

On the other hand, at receiver we use the other function as an interface to categorize and count the number of packets that are sent from different sources. As an example, we use the click-install script to start the function *sink.click,* which categorize the packets from different sources and count the number of packets received from each source.

2.3 Scenarios and examples of measurements.

This section contains one predefined scenario for a measurement. According to the parameters defined, the data flow is generated in the network. After all, the measured data is analysed by StreamAnalyzer. Plots and numerical statistics are generated providing the timing results of the analysis.

2.3.1. Scenario 1

These are the parameters for the packet flow generation, which define flow propieties as size of the traffic packets generated and amount of traffic to generate:·

Rate	2027
Limit	100.000
Length	61
MAC_O	00:0C:F1: CD:42:BE
SINK	192.168.0.1
Frame size	1500
MAC_D	00:11:11:0E:7E:80
DstIp	192.168.0.2
DPort	1500
Checksum	true
Interval	1

This data is stored by a Click program, which creates the desired traffic.

Once the previous click program is executed for generating the traffic, the packet capture is started by pressing the button Start in the Measurement Card's GUI presented in the previous point. The resulting file containing the capture needs of several format conversion processes until the file is in *.csv* format. Finally it is analysed using StreamAnalyzer, obtaining the following results: plots with the forwarding time and numerical statistics.

· The analysis of the timing and traffic parameters is out of scope.

The hosts involved in this measurement are described in the *Figure 3*. In the next figure, *Figure 6* we see the traffic flow generated from *h1* to *h2*. As it is noticeable, the forwarding time is about 8 microseconds at the beginning, because the other flow from *h3* to *h2* is not started yet. So after a while the forwarding time increases. It seems there should be some collisions and queuing at the switch because of the two flows (*h1* and *h3* acting as a traffic generators and *h2* as a sink), and at the worst case it reaches about 15 microseconds.

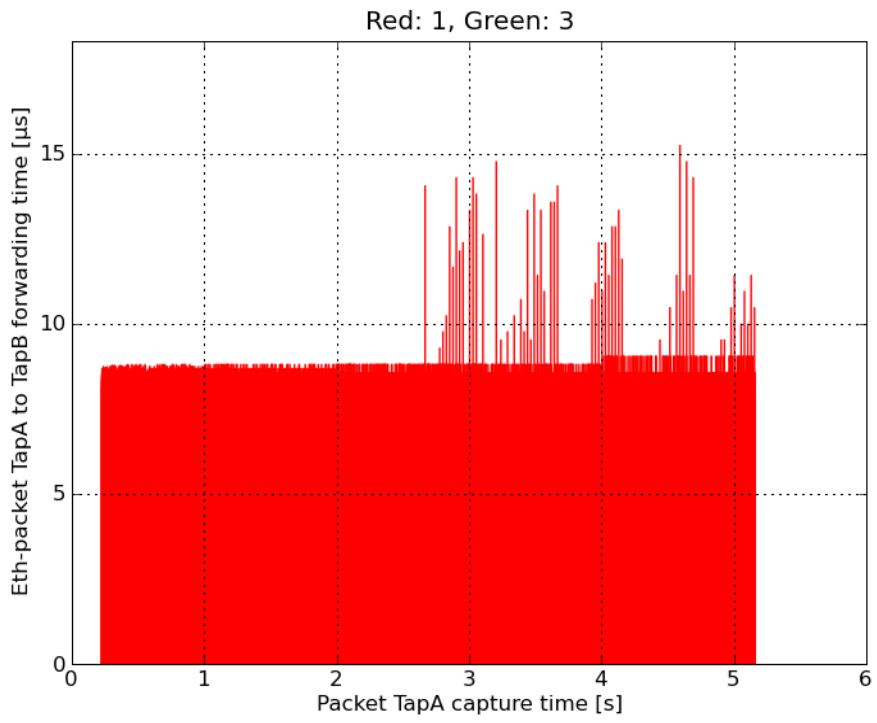


Figure 7: Flow from h1, forwarding time

The numerical statistics about forwarding time provided by StreamAnalyzer are as followed:
Forwarding time statistics:[·]

```
maximumFT = 15.2587890625 µs
averageFT    = 8.3169497884 µs
minimumFT  = 7.6293945312 µs
maxJitter      = 7.6293945312 µs
iBurstMaxFT  = 8849
iBurstMinFT  = 3763
iPacketMaxFT = 0
iPacketMinFT = 0
```

[·] The analysis of the obtained timing results and parameters is out of scope.

In *Figure 8* is shown the plot that contains the serialization pattern at maximum forwarding time. It shows up that certain packets from *h1* and *h3* are serialized at same time in this period, and that seems to make a collision, which might increase the forwarding time in the switch.

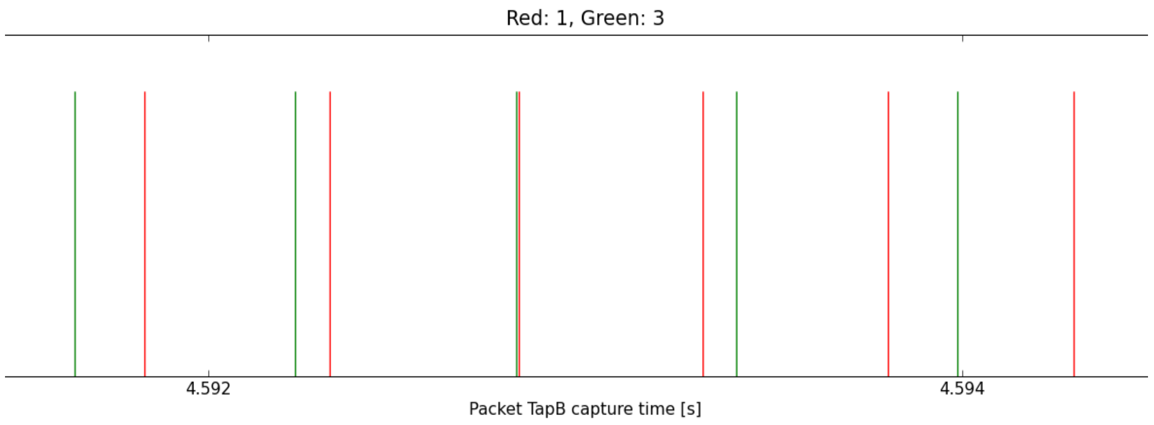


Figure 8: Serialization pattern at max. forwarding time interval

In this plot is possible to observe in which time instant are delivered the packets of each flow at the maximum forwarding interval time. As it is shown in the *figure 3*, which illustrates the flow generation, there are two flows generating traffic: host *h1*, whose packets are captured in red colour, and host *h3*, whose packets are captured in green colour. All of them have as a destination host *h2*.

Figure 9 contains the plot in which is observed that there is not constant delivery of packets and the jitter varies.

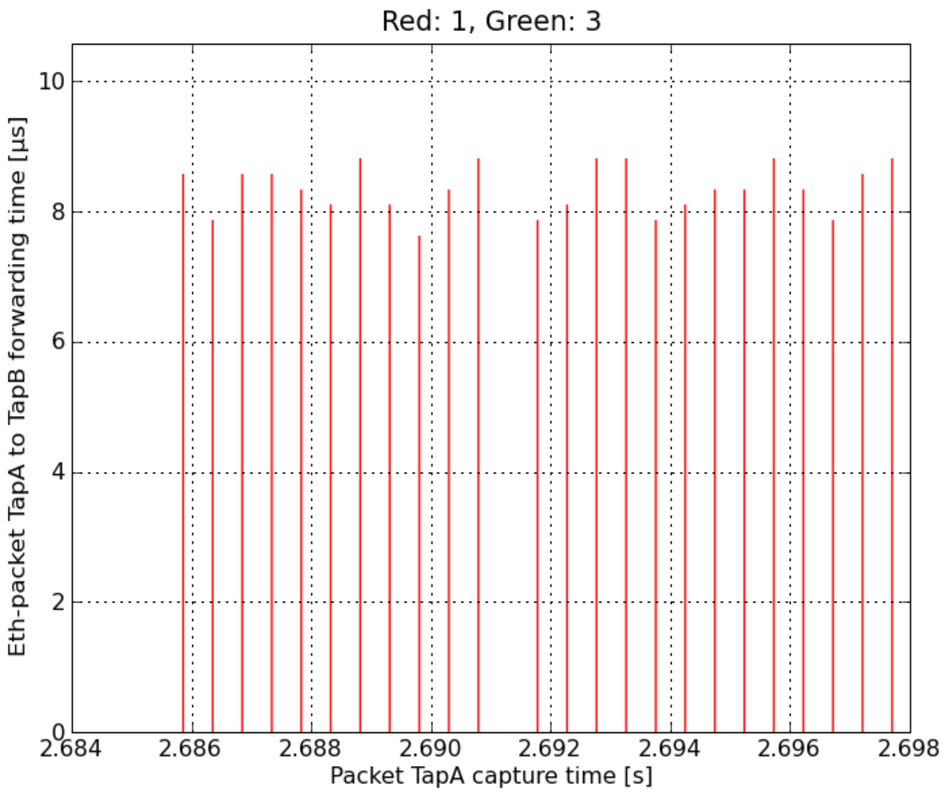


Figure 9: Jitter at max. inter-arrival time period

Finally StreamAnalyzer program provides numerical results about jitter timing analysis:[·]

```
Computing forwarding times ...

TapA stream interarrival time statistics:
maximumIT  = 988.2450103760 µs
averageIT     = 493.4716271237 µs
minimumIT  = 107.0499420166 µs
maxJitter      = 881.1950683594 µs
iBurstMaxIT  = 4996
iBurstMinIT  = 108
iPacketMaxIT = 0
iPacketMinIT = 0
```

[·] The analysis of the obtained timing results and parameters is out of scope.

3. Design & implementation

In this section are described the tasks needed for performing the automation of the system. The aim is to have at the end of this section a software and an architecture including new networks and devices in order to obtain a centralized control for generating measurements and obtaining the results without more need of human interaction than the needed for executing this tool. It should perform all the actions previously described in the distributed system for generating the traffic flows and provide the analysis results.

3.1. Make StreamAnalyzer work on Windows.

After analysing the possibilities for automatizing the system, we arrived to the conclusion that is better to have the program that captures the data and the program that analyses it in the same machine. It will avoid unnecessary movements of files with considerable size.

However, the issue is that the output file of the card containing the captured data, generated in a window system have to be moved in a manual way to another computer running on Linux in order to be analysed by the StreamAnalyzer program, which is compatible with Linux Systems. To solve this, we adapted the StreamAnalyzer program to work in Windows.

In this process we had to deal with some problems due to some functions do not behave in the same way in Linux and Windows, such as the function that provides the path and the function used to read from a file. Another issue is the way the path is defined in the program to find the desired files, and also the way they tokenize the names. So the program should be placed in a directory whose path does not contain spaces. Indeed, there should be no space in phrases used in python programs (python is used for generating plots in StreamAnalyzer).

Using the correct version of python and its libraries to plot the graphs was an issue, which is solved by installing python 2.6, *pylab* and other related programs.
On the other hand, to deal with the problems to set the system, we had to set the environment path according to the different programs, which deal with using StreamAnalyzer such as C++ compiler for windows, *python* and *pylab*.

This task seems a simple one at the first glance, but testing and debugging the program to find out the problems is not straightforward. After all we succeed creating the Windows adapted version, which provides same results and plots as the Linux version.

3.2 Create Hilscher-control software.

The way to interact with the card manually to start and stop capturing is done through the graphical interface, which is not suitable for automated systems. In this way, we decided to implement control software to interact with the card through C++ API, which is provided by Hilscher. In this program we will run start and stop capturing without using the graphical interface. So there will be no need of manual human interaction in this way.

In order to achieve this, the following functions defined in the API [4] are used:

- **netana_driver_information:** This function gets the driver´s information and its current state. Furthermore it says if there is some error accessing to the driver.

After getting access to the driver and checking that there is not any error it is necessary to define the parameters for the capture. The next functions are used:

- **netana_set_filelist:** This function activate capturing data into a *.hea* -file.
It defines the file path, file name, number of files to be created and file size.
- **netana_start_capture:** This function defines parameters to start capturing such as capture mode, capture reference time and etc.

Ring Buffer Mode
The latest version of the measurement card driver supports *ring buffer mode,* which allows to store captured data in more than one file. Each file size is fixed to 1 GB, so if the maximum amount of packets that fits into a file is reached, it will continue storing packets in the next file until the capture is finished or there is no more file to continue storing. In order to work in ring buffer mode, it needs to be specified before starting the capture. The function *netana_set_filelist* defined in the API [4] allows to define how many files have to be created. This will limit the maximum data that it can be stored. Also it is necessary to set up ring buffer mode in the function *netana_start_capture*, which starts the capture.

In this case it has been decided not to enable the ring buffer mode, which it will imply that StreamAnalyzer will needed to process more than one file and some extra time should be dedicated for this. On the other hand the capture size is up to 1GB and it has been considered as enough data to be processed in order to perform the analysis.

3.3 Avoiding format conversion

As mentioned, the StreamAnalyzer program is not designed to read the *.hea* file and it needs some conversion to be done. In this step, to reduce this inconvenient way of reading the captured data, we try to read directly from the binary file, *.hea*, in StreamAnalyzer program. The *.hea* file structure is described in the NetAnalyzer API [4]. As it is shown *in Figure 10*, each file has a header part, which is followed by captured packets.

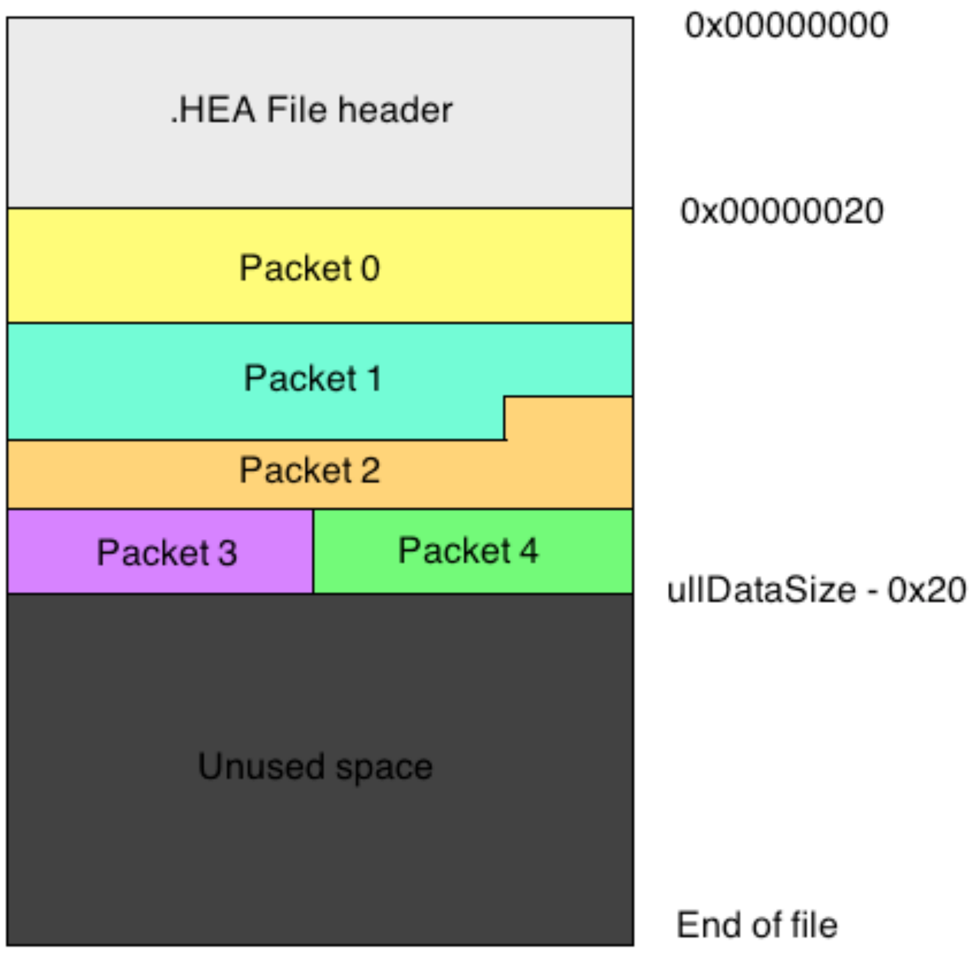


Figure 10: structure of the binary file with *.hea* format

The file header contains basic information of the capture such as the start time, file number and data size, which are useful to find out related capture files, the sequence of files, and where is the end of the captured data in file.

Sequentially, after the header part (which is 32 bytes), all the captured packets are placed until the data size is reached. In this part, the program should parse the packet. It is noticeable that each packet is padded to 32 bits and it should be considered to calculate the packet size, as it is not included in packet length. Each packet has a header, which contains the card port number, packet length and the time stamp.

After the header part, the packet fields should be extracted according to network headers structures like MAC header, IP header and UDP header. It should be mentioned that card driver such as packet header in big endian format and the data adds the network data and file header are in little endian format. So it should be considered to convert the network format, which is big endian. The first 14 bytes related to MAC header are discarded, and the following part is the IP header. In this part, due to the optional part of the IP header, we need to calculate the size of the IP header including the padding to know where the UDP packet starts.
Also, the protocol field is important, to be able to discard the packets which are not UDP. After all, the source and destination IP addresses are obtained from the IP header. The UDP packet starts after the IP header (including options). So the remaining fields including UDP source and destination ports are obtained in this part.
This process continues for all packets till the total data size (obtained from file header) is reached.

Limitations solved till now

After adapting StreamAnalyzer or being executed in Windows (3.1), creating the software that starts / stops the capture without needing interaction of the user through its GUI (3.2) and finally in this section (3.3) after reading directly from the *.hea* file, which is the format in which NetAnalyzer card's driver stores the capture, several manual steps can be omitted. It includes all conversions steps and the steps in which the user needed to move the file after the conversion to another machine in order to be processed by StreamAnalyzer.

At this point the capture is created without needing interact with any GUI, being stored in the local Windows machine that contains NetAnalyzer Card and being possible to run StreamAnalyzer directly from this machine and reading the original *.hea* file in order to obtain the desired results.

This implies an important improvement in order to achieve the total automation of the system and as well a reduction of the time needed for performing a capture and its corresponding analysis.

3.4 Create Coordinator software and network design

Once we have created the tools for initiate a capture without needing human interaction in the previous sections, avoiding to use GUIs, format conversions and manual movement of files, it is possible to design and implement a centralized control unit which will take care of all the actions needed for performing every step in this process in order to make a measurement.

It is also considered networks improvement and the adaptation of new devices, which need to be added to the system. Furthermore this process includes modifying and improving several of the steps that were executed as isolated before.

3.4.1 Create Coordinator software

This section is the main target of the project, to implement a centralized program to coordinate the whole process. This program called the Coordinator, executed in a Linux machine at the office network, will do all the tasks described below interacting with the different computers and transferring the results of the execution to the machine running the coordinator program. Therefore there is not needed any other manual contact with other machines to obtain the result of the analysis. The different tasks are:

- Connect to the switch to set the parameters according to the configuration file, which should be same as TCN-Analyzer configuration.
- Create traffic generator click scripts according to the flow parameters defined in configuration files for h1 and h3.
- Connect to different computers, such as traffic generators, sink and capture/stream-Analyzer machines.
- Start/Stop sink program on computer h2.
- Start/Stop traffic flows from h1 and h3.
- Start/Stop capturing with NetAnalyzer card, which is installed on Windows computer.
- Analyse captured data by StreamAnalyzer on Windows computer.
- Stop traffic generators and sink click scripts.
- Transfer results (numerical statistics and plots of the capture) from Windows machine to coordinator machine.

The coordinator computer is decided to be a Linux machine in office network, which is more convenient to start *SSH* connections from there to other Linux computers. Also, to make a connection to the Windows system, an *SSH* server is required to be already installed and listen to *SSH* clients. So, the FreeSSHd application [5] is used, which provides the *SSH* server for Windows operating system. In this way, the Linux machine can make a connection to Windows system, in order to execute the Capture and StreamAnalyzer programs remotely.

Choosing best programming language / architecture for the coordinator software

The coordinator program is decided to be **shell scripting** because this program is mainly connecting and executing commands on different machines which is simpler to handle this process in shell scripts, where it is possible to execute commands directly. Furthermore, **expect scripting** which is explained in [5] is useful for interacting with connected systems and execute programs through the SSH and SFTP connections remotely without manual interaction. For instance, when we create a SSH connection, the user does not need to type the password manually, because we can expect the system is requesting the password and send it automatically. In other words, the different outputs of the system can be expected and replied according to that specific output automatically.

In this way, there are several expect scripts that are used for connecting to other machines through *SSH* or *SFTP*, in other words connecting through some interactive protocol. One the other hand there is several bash scripts, in concrete the main program is a bash script which execute commands in the local machine and also other scripts such as the expect scripts that contact remotely with other machines and also executes another bash scripts that execute some sequences of commands for a same functionality, like updating the local routing table.

In this case *shell scripting is a better option* for our project than another alternative approaches such as it could be socket programming, in which a program should be written in each machine listening in one port for performing an action. Socket programming will not be interesting for our project, due that it would require start manually the server program previously wrote in C/C++ in each machine waiting for requests, and afterwards send messages to that servers in the coordinator. We look for performing the control of the system in a centralized way interacting only with one machine, in this case called the coordinator. Due to this it is obvious that socket programming is not the best option, and *shell scripting* fits more in our requirements.

Anther important point is the software maintenance / extension, in which we want to add some functionality or modify some part of the program. In case of having socket programming it would be needed to modify the program in each machine due that for performing one task more than one machine is needed.

On the other hand in shell programming is not needed generally to modify the code in each machine interacting in the process in case of any desired change. The control program is written in the coordinator and it includes all the actions performed for each computer, first the connection with the desired computer and later on the commands executed for the remote computer.

Therefore, it turns out that shell scripting is a better option in terms of system usage (not needed any manual interaction with other machines) and also in terms of maintenance and programming in which the main code that control the global interaction is placed in the coordinator program.

Next it is shown a list of tasks, which are executed sequentially in the main script that runs all the different components needed in the system for providing the desired functionally:

#Define variables
Define local variables that will be used in subscripts like time stamp for creating the folder with results named with the current data (year / month / day-time). Also some predefined directories are written in this part.

#add entries to local routing table
Here it is executed a bash script which need to login as a root for executing the commands for updating the routing table. It will add the necessary entries for finding the measurement networks and the control network in every machine of the office network where in which is executed this program. As it has been commented before, the only prerequisite is to use a Linux machine for running the coordinator program, which is obviously due that the coordinator program is written in shell-scripts. This is done with the bash script: *routingTable.sh.*

#Ping to different machines involved in the process
Here the program will *ping* the different machines involved in the process in order to update the switch and router entries. It will avoid to have later on ARP messages in which is trying to bind the requested IP address with the desired host MAC address which is used for transmission. This is done directly by the main coordinator program (placed at the office network). Furthermore, it is needed to invoke another *expect* subscripts, which connect via *SSH* to the sink hosts placed in a different network (measurement network) and *ping* from there the traffic generator hosts. This is executed by the subscript *pingFromSink.sh*, which receive as a parameter the IP address of the remote sink to connect with. Finally the selected sink host will ping all the traffic generators machines.

#Configure the Switch
Next it runs a expect script which connects with the *Westermo* switch which is connected in the measurement network. It will setup the configuration in the switch according to the parameters defined in a local file named: *switch_param.conf.* The expect scripts will read all the parameters from the file and it will intact with the switch sending the specific commands for interacting with the switch are defined in the Switch Management guide [6] , adding the previous obtained parameters and waiting the interactive (prompt) answer from the device, which indicates that the command have been received and it is possible to send the next one.
After this the program will wait during a short predefined time interval (2 seconds) while the switch will be applying the previous configuration

#Create Click scripts that generate traffic on h1 and h3,
In similar way than the previous step, in this part it is executed a *expect* script which connects to the traffic generators machines *h1* and *h3* through *SSH*, moving to the correct folder where are placed the click scripts that generate the traffic once they are being executed. Next, it will read as before the parameters from a predefined file named: t*raffic_config1.conf* and *traffic_config3.conf* , which contains the parameters that define the characteristics of the flow that have to be generated for the hosts *h1* and *h3* respectively.
In short, it contacts with the machine *h1* and *h3* where it writes down the desired click script, which will be executed later in order to generate the traffic flows. This *SSH* connections with *h1* and *h3* to do the described work are done in parallel.

# traffic generator CEC-boards set up (Added in point 3.5 as a new functionality)
In an initial step in this point we were connecting through *SSH* to the windows computer which has connected the boards and downloading the board's code each time into the boards after reading from a file the propieties of the desired flow. This was considered as a non-practical way of proceeding. Therefore, this procedure was redesigned, not being needed to read from any file the properties of the desired flow and not being needed to download the traffic into the card each execution time.

To achieve this, every card is constantly receiving requests. The coordinator executes a *java* program, which sends an UDP message to each board with the propieties of the desired flow, being possible different properties for each board.
Finally, every board that has received the propieties of the desired flow will be waiting for the order of start generating the traffic flow.

#Start sink
In this step the coordinator connects with the Sink machines through *SSH* in the same way than before using the expect script: *sink.sh.* This expect script execute the click script which will be listening for packets. Furthermore it is defined a time which is supposed that have to last the capture and after expiring this time it will stop the click-script which listen for packets and will turn down the *SSH* connection with *h2.*

#Start capturing packets in NetAnalyzer measurement card (windows machine)
This step is done executing the expect script: *capture.sh* . It connects through *SSH* with the Windows machine containing the NetAnalyzer card and executing the exec file, generated in the *section 3.2*, which starts capturing packets in the measurement card.

It is defined a predefined time value, which is supposed to last the capture. Therefore, after expiring this time, the capture is stopped (previously needed to press the button STOP.
Next it is executed StreamAnalyzer, which analyse the previous capture and calculate the timing plots and statistics. To finish, it will move all the resulting files to a folder named with the current data (year / month / day-time) in in which the capture was made.

# Make sure the capturing and sink programs are ready before generating the traffic
In this case it has been decided to wait during some seconds to make sure that the NetAnalyzer measurement card and the Sink machine are ready for receiving packets.

#Generate traffic flows in H1 and H3
Next it is run a expect script which connects to the traffic generators machines *H1* and *H3* through the specific control interface in the Control network for avoiding another packets interferences in the measurement network. The expect script that contains this functionality is named t*rafficGenerator.sh. These two connections with H1 and H3 are done in parallel.*

#Generate board traffic (Added in section 3.5 as a new functionality)
In this case it was requested to generate simultaneously traffic in every board. It implies that every board receives the *START* message at the same time instant. To achieve this, it connects through *SSH* to one machine connected directly to the measurement network, in this case *Mercury*. From this machine it is broadcasted the *START* message, which will be received for every card and the traffic generation will start. Done in subscript *broadcast.sh*.

#Transfer results to local machine

After waiting for the previous tasks have finished, it is executed an expect script named: *transferResults.sh,* which creates an *sftp* connection with the Windows machine and it will transfer to the coordinator machine the timing analysis and the statistics of the current capture. After this procedure the desired job is done, the capture has been started from a coordinator machine and it has obtaining the results of that capture without being needed any manual interaction with another machine.

3.4.2 Design separate Control and Measurement networks

As it is described in section 2, there is only one network domain for both measurement and control traffic so the coordinator makes connection to different computers in the same network as the measurement card wants to capture specific traffics, which is generated by H1 and H3. In this way, those control packets related to control network might interfere the desired traffic in the switch and change the measurement results for queuing and forwarding time.
Therefore, it is decided to have separate networks for each purpose:
One network is dedicated for measurement purpose. The second network is designed to control the different parts of the system, called control network. The third, the office network, is used by the persons at the office for connecting to Internet and to run the measurement program.

The measurement network (with IP: 192.168.0.0 netmask: 255.255.255.0) contains the netAnalyzer card, Westermo switch and *h1*, *h3* and *h2* hosts on their first network interface cards which is connected to this network. This part of network design is shown in *Figure 11*
So the traffic generated by *h1* and *h3* to *h2* are the desired flows, which will be captured by measurement card in this network.

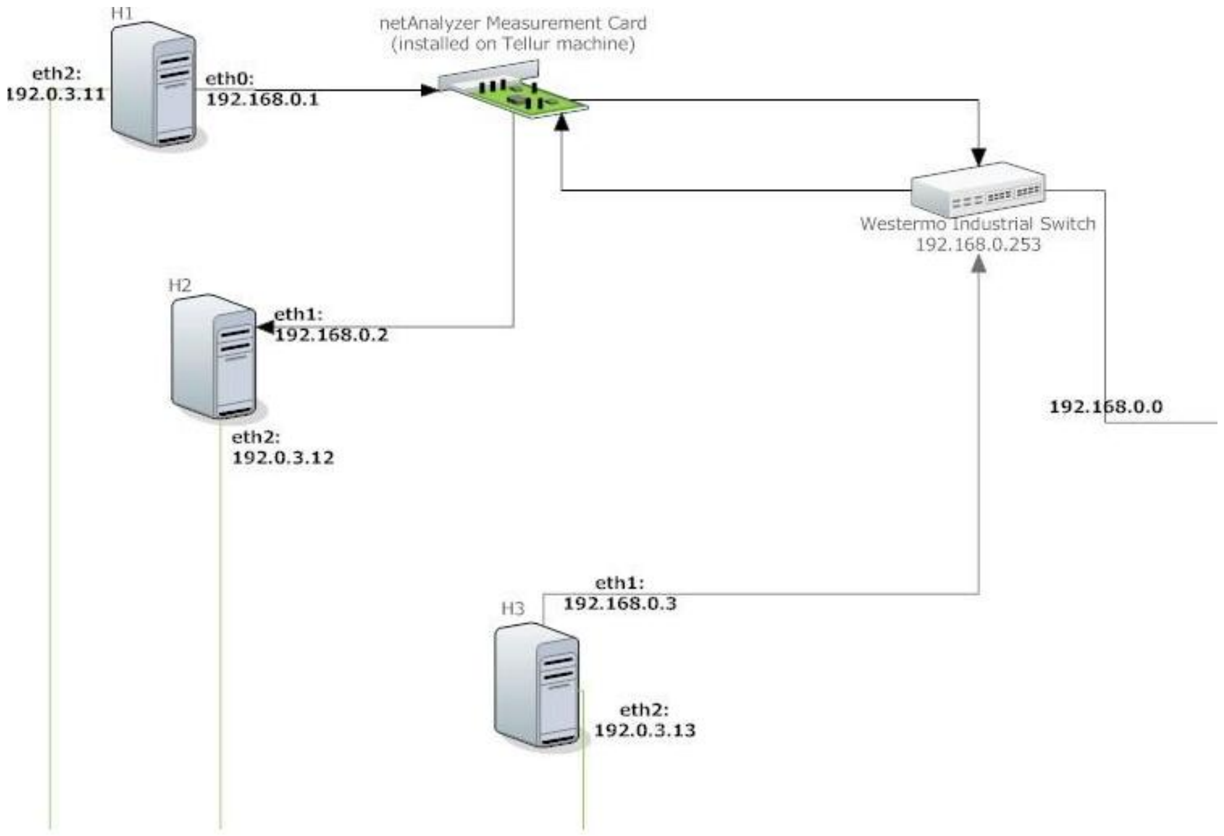


Figure 11: Measurement network

On the other hand, the control network (with IP: 192.0.3.0 netmask: 255.255.255.0) is supposed to be used by the coordinator to connect to different machines and run the control commands through this connection. This network also contains *h1*, h3 and *h2*, but with their second network interface which is assigned to 192.0.3.0 network. In this way the control flow is not passing through the switch of measurement network and won't make any interference in the measurement process.

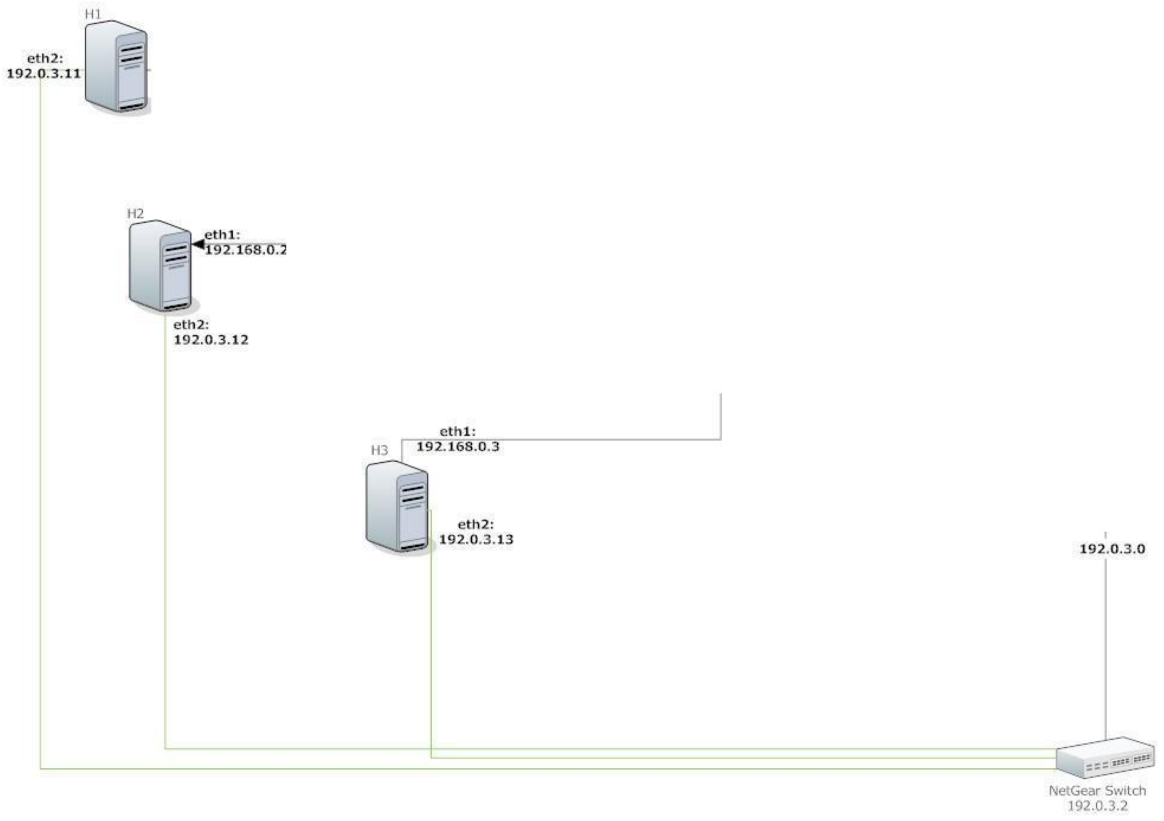


Figure 12: Control Network

The office network (with IP: 172.16.3.0 netmask: 255.255.255.0) , shown in *Figure 13* is used to connect to *Tellur* (Windows) and *Mercury* (Linux) machines. As described before, *Mercury* computer or any other Linux computer in office network can run the coordinator program to run the whole system process. *Tellur* computer is the one that runs netAnalyzer and StreamAnalyzer programs.

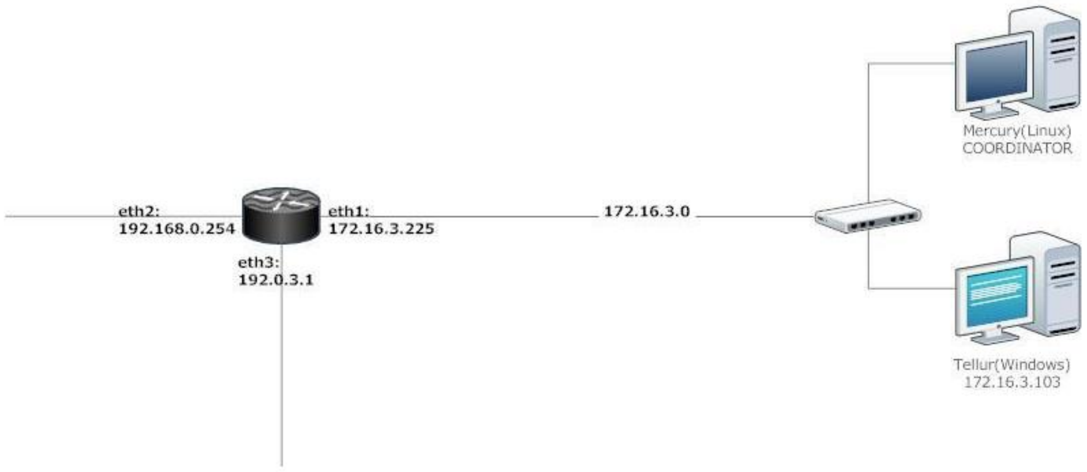


Figure 13: Office Network

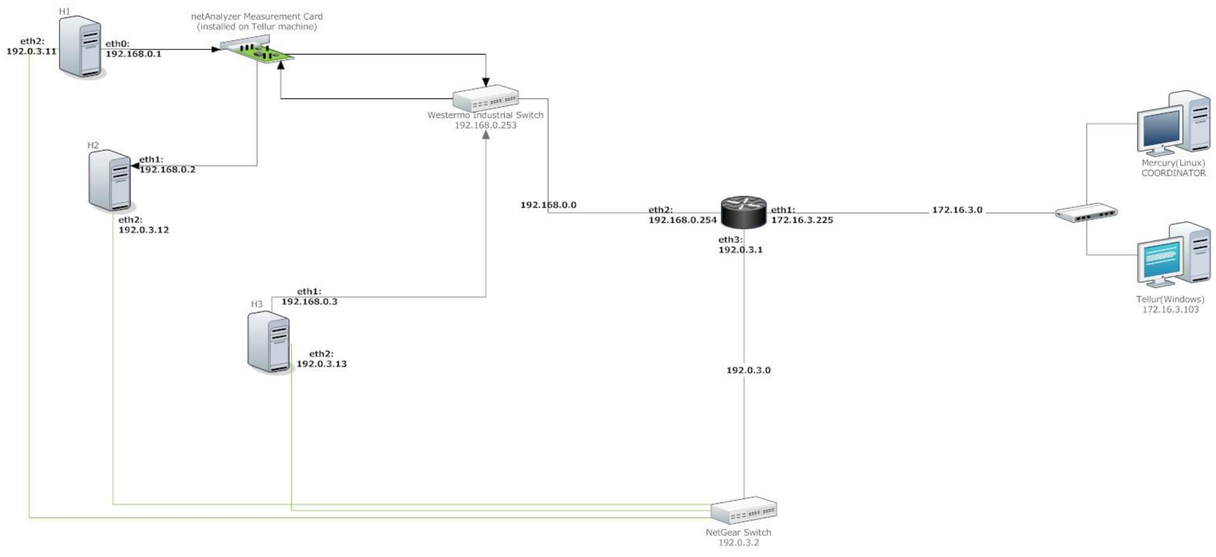


Figure 14: Complete Network Topology (higher resolution version at Appendix B)

The complete network design is found in *Figure 14*. As you can see the router *(r0)* is used to divide the network domains and connect them to each other. A better resolution picture is placed in Appendix B.

Routing table setup

Once the installation of the new network is done, including the assignation of new IP addresses for the new devices, it is needed to modify the routing tables of the different hosts involved in the transmission in order that the control packets are sent through the control network, avoiding any control packet to interfere in the measurement going through the measurement network.

Especially since the routing table of the hosts (traffic generators) *h1* and *h2* only contains entries for the measurement network and for the control network, some problem can happen to reply the requests from the coordinator placed in the office network.

When we connect through SSH from the coordinator computer placed in the office network, the messages arrive to *h1* and *h3* as it is expected. The issue comes with the reply of SSH packets in hosts *h1* and *h3*. The office network is not defined in their routing table due that it contains only two, one for the measurement network and one for the control network.
Therefore it is needed to set the default route (route used when the destination is not defined in the route table) as the router IP address of the control network interface. This will force the reply of the control messages go through the control network.

Finally when the reply messages arrive to the router they will be delivered to the office network and to the coordinator computer avoiding any interference in the measurement network as it was expected, separating the SSH control messages from the traffic in the measurement network.

3.5 Extend the architecture by adding CEC boards

Add board traffic generators

To extend the system to have different kind of traffic generators, CEC boards, which were used as CAN to Ethernet Converters, are programed to be used as Ethernet traffic generators. According to Ibrahim experiments that explained in [8], these boards can provide constant traffic flow, where the Click traffic generators are not able to provide that. Initially these boards were configured for generating CAN data traffic with [7], which was converted later on to Ethernet traffic. This has been omitted in this project and we generate Ethernet traffic directly programming the cards through a special framework provided by IAR [20].

Board specifications and Program

The testing board is a development kit made by IAR Systems [21]. It contains ARM-based microcontroller, LCD, different kind of ports for different applications, and USB port for download program on it and also provides power for the board. The driver is compatible with Windows, so the boards should be connected to Windows machine (*Tellur*) in our system.

To program the boards, the project is designed to assign specific IP to each board, and to be ready to send the UDP traffic when required. So the board is listening on a port to incoming packets and if it gets the UDP packet contains "START" message, the board starts the UDP traffic.

IAR-Systems provides a basic program for interacting with the board and providing a basic functionality. This program is written in C/C++ and was modified to be a CAN to Ethernet Converter by Ibrahim which is used in his Master thesis [8], but it this project, it is set to send UDP traffic with out CAN interaction, and also we adapted it according to our project requirements. According to our modification, the traffic configuration is not specified in the code, so it can be given according to each traffic specification and network design.

There were two approaches to do this step. First, the traffic specification parameters could be read from a file with the board program. In this way, the coordinator computer should edit the configuration file and transfer it to Windows machine, allowing the board to read the parameters from the file.
In second approach, the flow parameters are sent in a *UDP* packet with "SET" message. In this way the program can set the traffic according to the parameters before generate the traffic, being needed to download the code into the board each execution time. The first approach is not efficient. Each time the configuration file is changed the board program

should read the new parameters and download the code in every card, which takes some seconds. Furthermore, this does not allow assign different flow propieties to every board.
So finally, the followed approach in this project has been to download the program into the board just one time, being listening for requests. According to this way of proceeding, different traffic propieties can be sent to different cards. The cards will be waiting for a *SET* message, which will set the flow propieties. Later on, it will wait for a *START* message, which will trigger the flow generation.

As it has been described, the program listen to the incoming messages forever, and when it gets the *SET* message, the card set up the flow configuration propieties according to the values received in the SET message. Afterwards the *START* message is broadcasted to every board. This is done in order to allow every board start sending messages simultaneously, due that if the *START* message is sent sequentially, the boards might be operating in different phase.

Integrating boards into the Coordinator Program

The coordinator should be able to communicate with the boards in order to run the program and send the START message to start the traffic generation. The card is not designed to be able to connect through *SSH* or other communication programs, it is not possible to program it in these ways. The boards are connected through USB port to Windows machine, which is used to download the program and also power it.

As it has been described, once the program is downloaded into the boards they will be waiting for the reception of the *SET and START* message that will configure the flow and trigger the traffic generation.

In this way, we use Java socket programming on the Coordinator side, which communicate with the board on specified port sending the *SET* message containing the parameters and the START message. On the board's side, is used socket programming but in this case with C programming language due that the basic program provided for IAR for interacting with the boards is written in this language. This java program reads the flow propieties from the file in which the user has written them, sending the *SET* messages later on to each board.

Final topology of the System

After adding the CEC-boards to the resulting topology is shown in *figure 15*.

Here are highlighted the three different networks. The office network contains the different machines, PC and laptops that are used by everybody in the office. It has been decided that the coordinator program, the program that controls the whole system, can be installed in any machine connected to the office network (Linux dependant). This makes easier to use the system by different users at the office in their own computers.

It was decided to create a new network for sending the messages from the coordinator to the other hosts in the system for controlling and performing the different tasks of the system. The main reason for a different network is to avoid having control messages interfering in the measurement network.

Finally the measurement network contains the different units that will create traffic and measure it. It includes advanced switches, a real time measurement card, CEC-boards that will generate traffic and certain hosts that can generate or receive traffic (sinks).

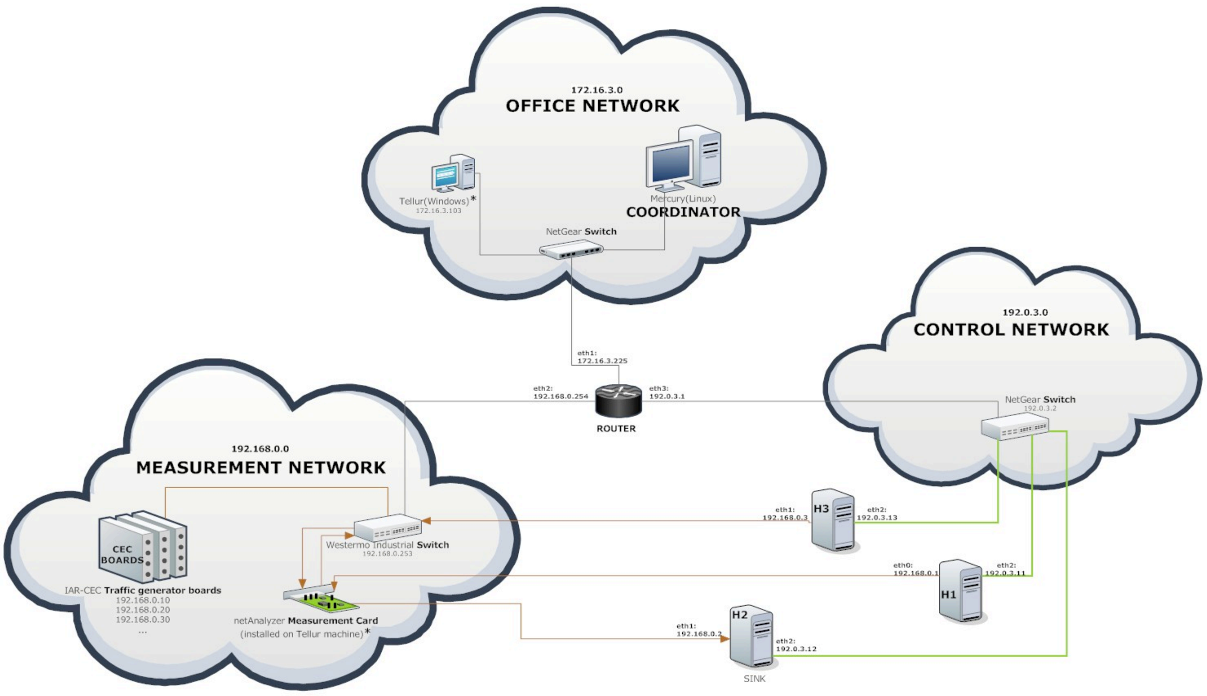
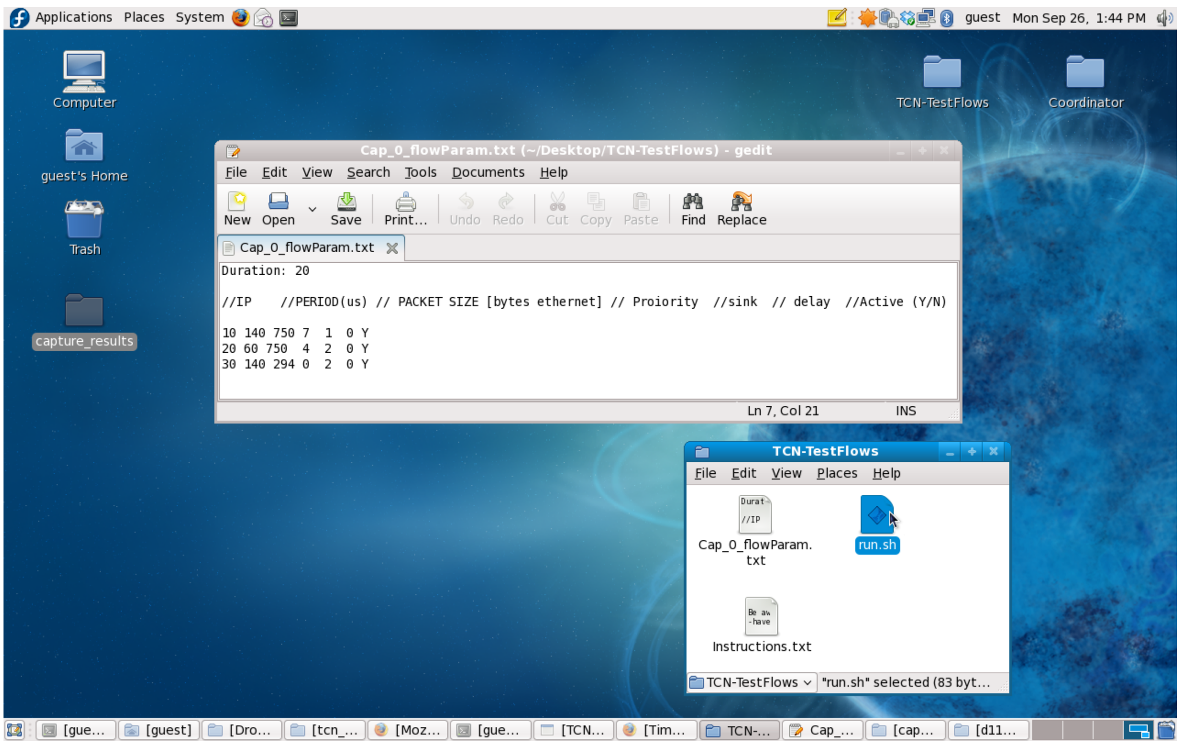


Figure 15: Different networks interacting int he Measurement System

4. Execution and Results

The aim of the project has been to create a centralized control environment that allows the user to define the required traffic flows, run the system and obtain the results of the measurement some seconds afterwards without being needed to interact manually with any other computer of the system. Here is presented the developed tool result of this project.

In *Figure 16* is shown the system interface. The coordinator program that will launch the measurement consists of two folders. All the files that are the heart of the system are placed in the 'Coordinator' folder, which includes the main program and all the secondary scripts created in order to perform specific functions.
The user just need to access to the folder called *TCN-TestFlows (renamed later on to TCN-Measurement tool)* where there is a text file in which the user needs to define the flows to be generated and their propieties.



```
Duration: 20

//IP    //PERIOD(us) // PACKET SIZE [bytes ethernet] // Proiority  //sink  // delay  //Active (Y/N)

10 140 750 7  1  0 Y
20 60 750  4  2  0 Y
30 140 294 0  2  0 Y
```

Figure 16: TCN-Measurement tool, execution environment

Once the user has written the flows propieties, it is possible to execute the system by running the file "run.sh" in the same folder. It is possible to run it by doing double click with the mouse on the file or executing it directly from the terminal. In this last case will be possible to observe the out prints that the system generate, which can help the user to make an idea which tasks are being executed in each moment and in which part of the system.

When the execution has finished the results are available in the folder ”capture-results” in the desktop.

As it has been said, the results of the capture analysis are provided in the folder ”capture-results”, in which a different folder is created for each execution of the system. Each folder is named with the date and time in which the execution has been done.

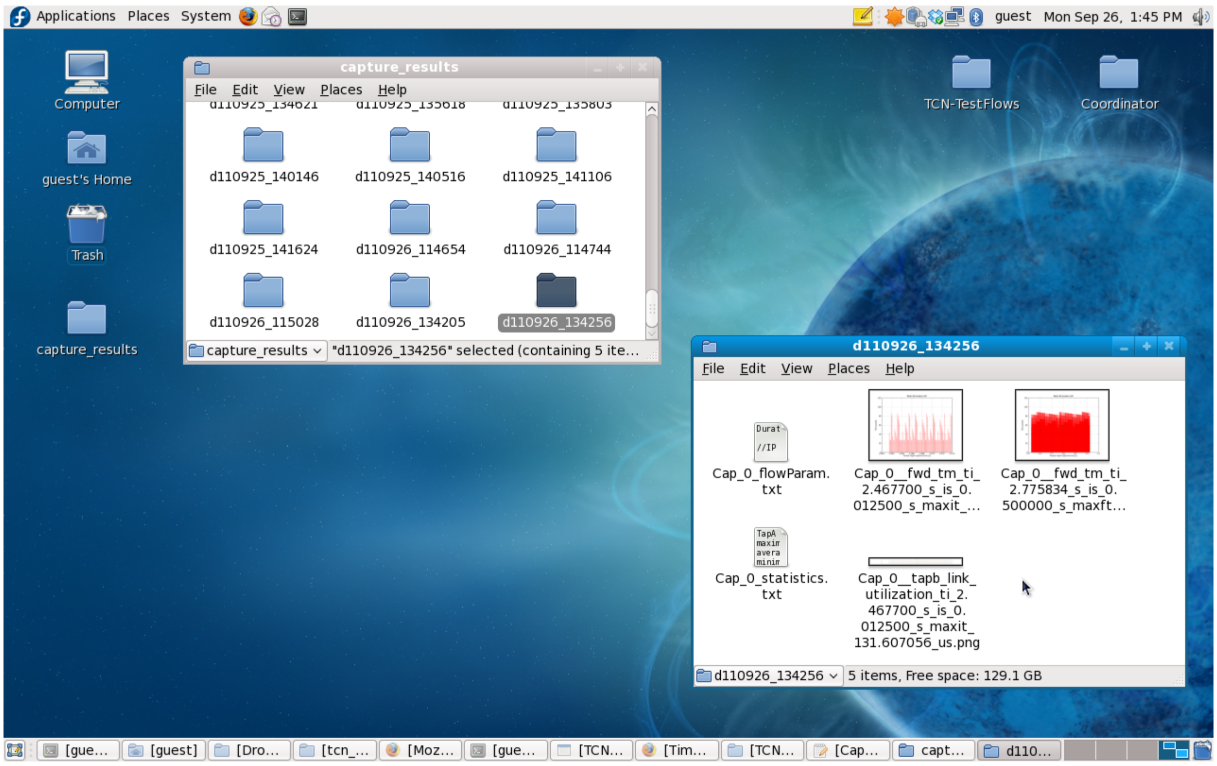

Applications Places System
guest Mon Sep 26, 1:45 PM
Computer
guest's Home
Trash
capture_results
TCN-TestFlows
Coordinator
capture_results
File Edit View Places Help
d110925_140146
d110925_140516
d110925_141106
d110925_141624
d110926_114654
d110926_114744
d110926_115028
d110926_134205
d110926_134256
capture_results
"d110926_134256" selected (containing 5 ite…
d110926_134256
File Edit View Places Help
Cap_0_flowParam.txt
Cap_0__fwd_tm_ti_2.467700_s_is_0.012500_s_maxit_…
Cap_0__fwd_tm_ti_2.775834_s_is_0.500000_s_maxft…
Cap_0_statistics.txt
Cap_0__tapb_link_utilization_ti_2.467700_s_is_0.012500_s_maxit_131.607056_us.png
d110926_134256
5 items, Free space: 129.1 GB
[gue… [guest] [Dro… [tcn_… [Moz… [gue… [TCN… [Tim… [TCN… [Cap… capt… d110…

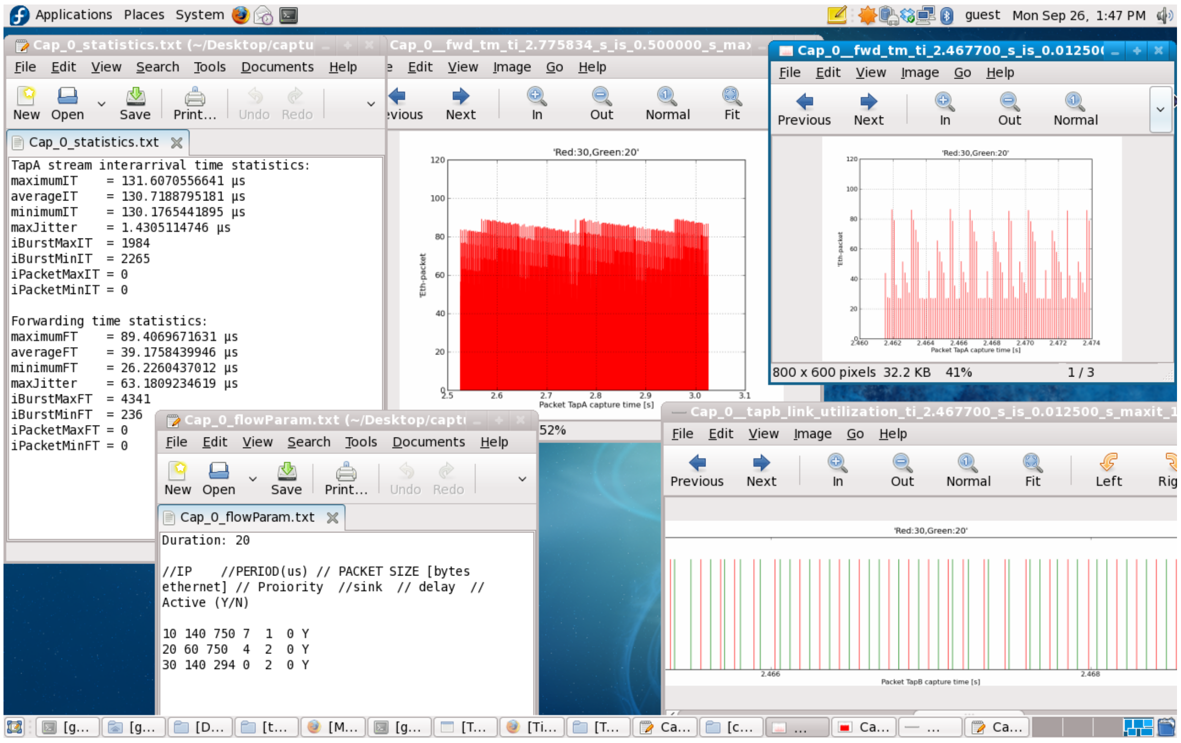
Figure 17: Results of the measurement

The results consist in different files that define the characteristics of the flows and several plots that are the result of performing the timing analysis of the generated traffic. As well one text file containing the statistics (timing results) that are possible to be observed in the plots is created for simplifying the task of comparing the results of the execution.

Figure contains the files created as a result of an execution which are stored into a folder with the time in which was performed the measurement.

It contains the file in which the user defined the flows characteristics, so it is possible to compare the timing analysis provided for a certain flows setup, with different flow number, packet size, priorities and period.

Furthermore three plots are created in which it is possible to observe which is the maximum forwarding time in the system, in other words the maximum forwarding time it takes the system to deliver the packets. These results can be observed as well in a statistics file that includes some extra information.


Applications Places System
guest Mon Sep 26, 1:47 PM
Cap_0_statistics.txt (~/Desktop/captu
Cap_0__fwd_tm_ti_2.775834_s_is_0.500000_s_max
Cap_0__fwd_tm_ti_2.467700_s_is_0.01250(
Cap_0_flowParam.txt (~/Desktop/capt
Cap_0__tapb_link_utilization_ti_2.467700_s_is_0.012500_s_maxit_
File Edit View Search Tools Documents Help
New Open Save Print... Undo Redo
Previous Next In Out Normal Fit Left
'Red:30,Green:20'
Eth-packet
Packet TapA capture time [s]
Packet TapB capture time [s]
800 x 600 pixels 32.2 KB 41% 1 / 3
52%

```
TapA stream interarrival time statistics:
maximumIT    = 131.6070556641 µs
averageIT    = 130.7188795181 µs
minimumIT    = 130.1765441895 µs
maxJitter    = 1.4305114746 µs
iBurstMaxIT  = 1984
iBurstMinIT  = 2265
iPacketMaxIT = 0
iPacketMinIT = 0

Forwarding time statistics:
maximumFT    = 89.4069671631 µs
averageFT    = 39.1758439946 µs
minimumFT    = 26.2260437012 µs
maxJitter    = 63.1809234619 µs
iBurstMaxFT  = 4341
iBurstMinFT  = 236
iPacketMaxFT = 0
iPacketMinFT = 0
```

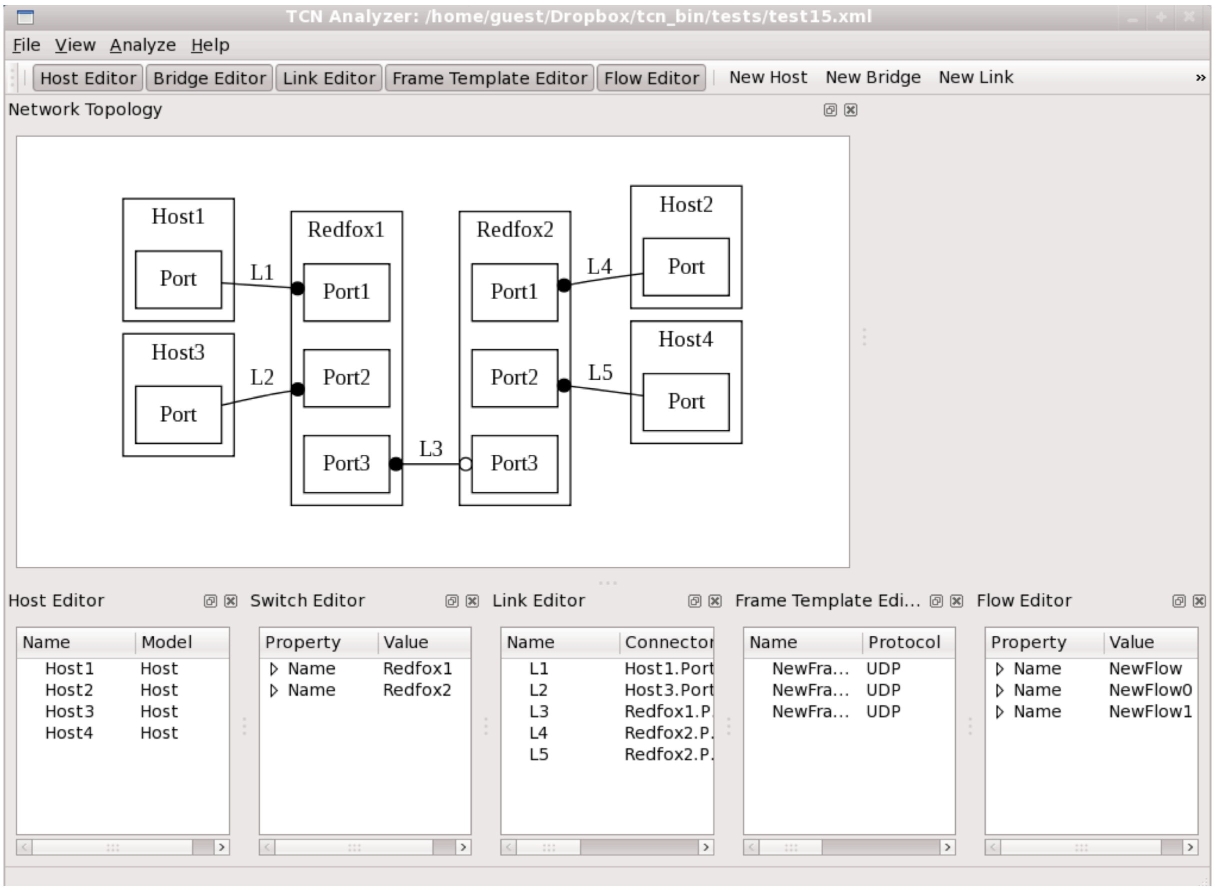
```
Duration: 20

//IP    //PERIOD(us) // PACKET SIZE [bytes
ethernet] // Proiority  //sink  // delay  //
Active (Y/N)

10 140 750 7  1  0 Y
20 60 750  4  2  0 Y
30 140 294 0  2  0 Y
```

Figure 18: Resulting analysis performed by StremAnalyzer

As it seen it has been avoided any extra manual interaction. Just is needed to run the main program, which will take care of executing and processing the local outputs that before needed to be take in consideration manually. After this the results will show up at the same machine where we executed the program.

Comparing the measurements results with the TCN-Analyzer prediction.

As it has been described before, TCN-Analyzer is the main tool or product developed at the company Time Critical Networks AB. This tool is an interactive program in which the user defines the components of system to be analysed and the flows desired in it. It takes advantage from the scheduling algorithms developed at the company and improved by Meng Liu's thesis project [19].



Name	Model
Host1	Host
Host2	Host
Host3	Host
Host4	Host

Property	Value
▷ Name	Redfox1
▷ Name	Redfox2

Name	Connector
L1	Host1.Port
L2	Host3.Port
L3	Redfox1.P
L4	Redfox2.P.
L5	Redfox2.P.

Name	Protocol
NewFra...	UDP
NewFra...	UDP
NewFra...	UDP

Property	Value
▷ Name	NewFlow
▷ Name	NewFlow0
▷ Name	NewFlow1

Figure 19: TCN-Analyzer GUI

Figure 19 shows a non-final version of the TCN-Analyzer GUI in which it is possible to define the network propieties, here it is shown that four hosts have been defined and two Redfox switches. The traffic characteristics have been defined in the lists above the picture.

When TCN-Analyzer performs the analysis, it predicts the maximum forwarding time, in the worst case. This is what companies and potential customers using the TCN-Analyzer tool are interested in.

Goal of the comparison

On the other hand the aim of this thesis has been the development of the Centralized Measurement Control described in previous sections.
The centralized measurement tool developed in this project, TCN-Measurement, preforms measurements in the Distributed System presented before according to the traffic defined at the TCN-Analyzer tool and performs a timing traffic analysis.

The goal of this is to compare the timing prediction provided by the TCN-Analyzer tool and the timing results obtained with the TCN-Measurement tool in order to be sure that TCN-Analyzer tool provides an adequate time prediction that customers can trust.

Therefore TCN-Analyzer is a tool that provides a prediction of the maximum forwarding time in which the packet flows are delivered to its destination.
For performing this action TCN-Analyzer takes in consideration worst-case response time analysis. More information about the TCN-Analyzer tool and the algorithms used for calculating the maximum forwarding time can be found at Meng's Master thesis [19].

5. Future Work

Every system is always ready for improvements or new tasks and functionalities that could complement it. This system is not different and several improvements and new tasks have been performed at the last period when the systems seemed to be already finished.
One point to make improvements is to achieve major integration of the TCN-Analyzer with the TCN-Measurement system.

As it has been described, TCN-Analyzer performs a prediction of the worst forwarding time after the user has created in their graphical interface the virtual model of the network to analyse. Next, in order to verify the correctness of the prediction it is needed to run the TCN-Measurement tool described in the previous part developed in this project.

The first improvement is to consider how both programs are executed, not existing interaction between them.
Due that the user defines the flows propieties at the same time that creates the model of the network using the GUI of TCN-Analyzer, it should be possible to send this information to the measurement system without being needed the user to rewrite the flows propieties in a text file as it is described in *section 4*.
Another improvement could be to plot in the same graph the prediction made by TCN-Analyzer and the value calculated by TCN-Measurement system of the worst forwarding time.

Regarding to the *research*, this pretends to extend *section1, Introduction*, where it is described the main propieties of CAN and Ethernet as a protocols for transmitting data into a link. Furthermore it is discussed why industry is interested in applying Ethernet technology when they currently use CAN bus commutation and how TCN-Analyzer takes advantage of the lack of tools that perform timing analysis, design and verification of Ethernet networks.

As it has been described, most of applications in industry operate with CAN and the programs currently available for design and verify networks in these time critical systems operate using CAN networks. Thus this tool pretends to be leader in the promising technology of using Ethernet in time critical applications such automotion and controls for vehicles.

Next there are three products that try to solve same or similar problems than TCN Analyzer:

- Symtavision [22]
- INCHRON [23]
- Real-Time at Work (RTAW) [24]

However they use different techniques like simulation instead of *worst-case response time analysis* that is used in TCN-Analyzer. In the case that they use the same technique, they are targeting CAN networks and/or task scheduling in ECUs (Electronic Control Units) in vehicle networks instead of Ethernet.

As it has been described in *section 1*, TCN product, TCN- Analyzer focuses on creating a network model by means of an interactive GUI, being able of performing afterwards the analysis and prediction of the timing constraints of the virtual Ethernet network created.

Many companies and institutions are already showing interest about adding Ethernet to their time critical systems, creating research projects in order to validate that standard Ethernet can be used in this time critical applications without major changes.

In fact, as described in *section 1*, research project of BMW conclude that IP-Ethernet could suit well the real-time requirements even for safety-critical applications [9].
Furthermore, TCN and CPAC, a company of Volvo group have already verified the possibility of replacing partially or completely CAN bus with Ethernet without major changes. Another company that shows interest into the field is Siemens Automation, the largest company in the world in this segment, has recently started a project intending to verify Ethernet usage in time critical applications [16].

There is no doubt that the usage of Ethernet in time critical applications will be a reality not far away in time, and TCN-Analyzer product together with TCN-Measurement tool developed during this project for testing and verifying TCN-Analyzer's predictions will be well placed in the new market of this applications.

6. Conclusions

This report claims to give an insight into the current protocols used for transmitting data into a communication link from the point of view of time critical applications.

It points out the growing industries' interest in Ethernet protocol for time critical applications. It provides better throughput and do not have certain limitations such as the previous ones described about CAN.

TCN-Analyzer tool allows designing Ethernet networks and verifying the time correctness of time critical applications giving a prediction of the maximum forwarding time for a flow in the network.
During this project has been developed TCN-Measurement tool. It create traffic flows and measure the time it takes to the flows to reach the destination, being able to compare the prediction done by TCN-Analyzer with the real measurement performed by TCN-Measurement tool.

At the end of this project, it has been possible to perform a measurement just executing the tool developed in this project, obtaining the timing results in few seconds. On the opposite, in the previous system version it was needed to execute manually several tasks on different computers and moving by manual methods the processed results between machines involved in the process. This took several minutes for obtaining the timing results, which TCN-Measurement tool obtains in several seconds without needing any interaction except the execution of the start command.

To conclude, comment about the good commitment of the development process. It was planned to develop and perform the automation of at least one part of the measurement process and finally at the end of the project we achieved the automation of the complete measurement process.

7. Terminology and abbreviations.

This section explains different abbreviations and protocols used during this document [25].

- **SSH:** Secure Shell. Allows to access to a remote terminal in a secure way.
- **Ping:** Utility that uses ICMP protocol for testing the reachability of a host in the network.
- **Python:** Programming language.
- **Pylab:** numeric computation environment with plot functions.
- **SFTP:** SSH File Transfer Protocol.
- **ARP:** Address Resolution Protocol.
- **MAC address:** Media Access Control Address.
- **Switch:** Network device that connects networks segments.
- **Driver:** Program that allows higher-level programs to interact with a hardware device.
- **Routing table:** table stored in a network device that lists the routes to network destinations.
- **UDP:** User Datagram Protocol.
- **USB:** Universal Serial Bus.
- **TCN:** Time Critical Networks.
- **TCN-Analyzer:** TCN’s product**.**
- **StreamAnalyzer:** TCN’s analysis program, which obtains the timing results.
- **IP:** Internet Protocol.
- **LCD:** Liquid crystal display.
- **C:** Programming language.
- **C++:** Programming language.
- **Java:** Programming language.
- **CEC:** CAN to Ethernet Converter.
- **CAN:** Controller Area Network.
- **GUI:** Graphical User Interface.

8. References

[1] Björn Andersson and Jonas Lext, *Guaranteeing Hard Real-Time Communications Over Standard Ethernet Networks*
[2] Eddie Kohler, *The Click Modular Router*, Ph.D Thesis
[3] *Hilscher Gesellschaft für Systemautomation mbH, User Manual netANALYZER NXANL 50-RE*
[4] *Hilscher Gesellschaft für Systemautomation mbH, Driver Manual netANALYZER API Windows 2000/XP/Vista/7 V1.3*
[5] Expect homepage: http://expect.sourceforge.net/
[6] *Westermo OS Management Guide. RedFox Series Wolverine Series 6101-3201. Version 4.2.0-15841*
[7] PCAN-View: http://www.peak-system.com/Product-Details.49+M5d31ea5c82c.0.html?&L=1&tx_commerce_pi1[catUid]=10&tx_commerce_pi1[showUid]=32
[8] Mohammad Ibrahim, *Ethernet in Steer-by-wire Applications,* M.Sc., KTH Royal Institute of Technology, July 2001.
[9] Industrial Ethernet advisory group: http://www.industrial-ethernet.org/ethernet-protocols-technologies/a-universal-network-for-in-car-control-systems/
[10] Wilfried Voss, President esd electronics, Inc USA. *The Future of CAN / CANopen and the Industrial Ethernet Challenge*: http://www.rtcgroup.com/whitepapers/files/TheFutureofCAN.pdf
[11] TCN news post: http://www.timecriticalnetworks.com/index.php?option=com_content&view=article&id=4&Itemid=5
[12] Ethenet standard IEEE 802.3: http://standards.ieee.org/about/get/802/802.3.html
[13] Gigabit Ethernet: http://www.cisco.com/en/US/tech/tk389/tk214/tech_brief09186a0080091a8a.html
[14] Pcap file format: http://www.fileinfo.com/extension/pcap
[15] Wireshark tool page: http://www.wireshark.org/
[16] Workshop event: http://www.timmo-2-use.org/events/OpenWorkshop2011.html
[17] CAN protocol: http://www.can-cia.de/fileadmin/cia/specifications/CAN20A.pdf
[18] Jeff Nowling, Industrial Automation Engineering consultant. *How fast is Ethernet*: http://www.cross-automation.com/TechCenter/AppNotes/Supplier/Ethernet.pdf
[19] Meng Liu, *Response Time Analysis of Ethernet Flows with Fixed Priority - Evaluation and Improvement of TCN Analyzer Scheduling Analysis Theory*. M.Sc. Chalmers University.
[20] IAR framework: http://www.iar.com/en/Products/IAR-Embedded-Workbench/
[21] IAR Systems: http://www.iar.com/en/
[22] Symtavision, http://www.symtavision.com/
[23] Inchron, http://www.inchron.com/
[24] Real Time at Work, http://www.realtimeatwork.com/
[25] Wikipedia, http://www.wikipedia.org/

9. Appendix

Appendix A - Measurement Setup before the thesis project

This section contains specific setup information that was only valid for the initial network architecture previous to the start up of this project.
After the development of this project this information is not needed anymore, being all the tasks automated and the network architecture updated.

Boot lab network
Start the machines r0, h1, h2, h3, the voltage source, which drives the Redfox switches, and the netGear switch, for control network.

Click
The hosts in the lab network uses the open source project Click, which enables them to function as, e.g., a flow source.

Route to the lab network
TCN's office computers are attached to the Chalmers office network, which is shared with all the other incubator companies on sixth floor. The office network address is 172.16.3.0/255.255.255.0.
TCN's lab network on the other hand, contains two networks; measurement network and control network. Measurement network has the address 192.168.0.0/255.255.255.0 and Control network has the IP address 192.0.3.0/255.255.255.0. The three networks are connected via one of the Click-machines, r0, which acts as a router. To tell the computer how to access the lab network use either the route command or a system tool:

Route Command
```
route add -net 192.168.0.0 netmask 255.255.255.0 gw 172.16.3.225
route add -net 192.0.3.0 netmask 255.255.255.0 gw 172.16.3.225
```

Now, this step is done in script *routingTable.sh,* so, it is not needed to do this step manually.

Set the Internal Clock
The internal clocks of hosts h1 and h3 will probably be wrong at boot time. This leads to a compiler warning when compiling the Click module and possible that not all files are recompiled. The right time can be set with the command date:
```
date - set "2010-04-20 15:32:00"
```

File Exchange
If one needs to transfer, for example, a new click-script or Click elements from argon, the scp command could be used:
```
scp -r tcnflowsrc.* username@192.168.0.1:/home/username/click-1.5.0/elements/linuxmodule/
```

Click scripts and measurement procedure
The following section describes how the initial measurements in setup have been performed and some experiences.

The Click-script that is used to generate traffic during the tests have been called FastUDPSource.click on both h1, h2, and h3. In these, we set the rate parameter high in h3, so it works as "stress flow". The script sink.click on h2 acts as a sink for both the flow from h1 and h3 and the number of packets received from each flow can be read by running cat:

cat /proc/click/count_from_h1/count
cat /proc/click/count_from_h3/count

Log in with SSH to the sources h1 and h3. FastUDPSource.click can be started on them without the SSH connection hangs.

Download filter in NetAnalyzer by selecting **File** » **Filter Settings** and select **Load**. Select the file C:\Documents and Settings\All Users\Documents\net_analyzer_filter_reject_h3.nff. This filter rejects all packets coming from host h3 by matching against the 'Source MAC Address'.

Sink h2
h2 should be operated via the directly connected keyboard and monitor because SSH connection hangs when running click-script on this computer:
• Login as root.
• Start the click by issuing the command click-install /root/click/sink.click
• Go to the folder /click
• We can now check how many packages the sink accepted by giving the command cat count_from_h1/count count_from_h3/count

NetAnalyzer on Tellur
Tellur is the Windows computer with the Hilscher NetAnalyzer card installed.
• Start the application NetAnanlyzer from the Windows Start Menu
• Ensure that all tap ports are active
• Start NetAnalyzer measurement by clicking **Start**
• Start the packet generator scripts (see below)

Source h1
By running the click script through the bash script run_source, we can avoid the SSH connection hanging forever. This script uninstalls the click-script after 30 seconds after it finished running.

On h1: drive click-install FastUDPSource.click
On h3: drive click-install FastUDPSource.click

Start at NetAnalyzer then convert, double-click the file to start wireshark.

- Wait until all / enough packets have passed. Press Stop
- The NetAnalyzer driver routine now saves the measurement in the file C:\Default.hea
- Press *Convert* in NetAnalyzer GUI to translate this file to the format *.pcap* with a suitable filename.
- Double-click the new *.pcap* file. This will start *wireshark*, which shows all the captured packets during the measurement.
- In order to allow StreamAnalyzer to process the capture file, we must translate the file again to the format *.csv* (Comma Separtated Values). In *wireshark*, select **File** » **Export** » **File**. Enter the same filename as the *pcap* file, but change the extension to *.csv*. Select file format ***.csv***.

Misc
- To edit scripts use for example the application nano: nano FastUDPSource.click
- A click-install scripts can be stopped with the command click-uninstall
- Through the tests that found that the Click element FastUDPSources can send packets with the length parameter is <= 1518th In all cases took Basin in h2 receive as many Ethernet frame sent. Should actually theoretical upper limit be 1514 bytes. According to a note in the Click script will, however, the measurement card to protest when LENGTH> = 1515, which agrees with the theory. Could be of interest to some time to examine where the four extra switches will be added. Could it be that the extra bytes are inserted into the header to match the second Ethernet headers than the basic version?

Network Shutdown
SSH root@192.168.0.1 shutdown -h 0
SSH root@192.168.0.2 shutdown -h 0
SSH root@172.16.3.225 shutdown -h 0

Appendix B - Network Architecture

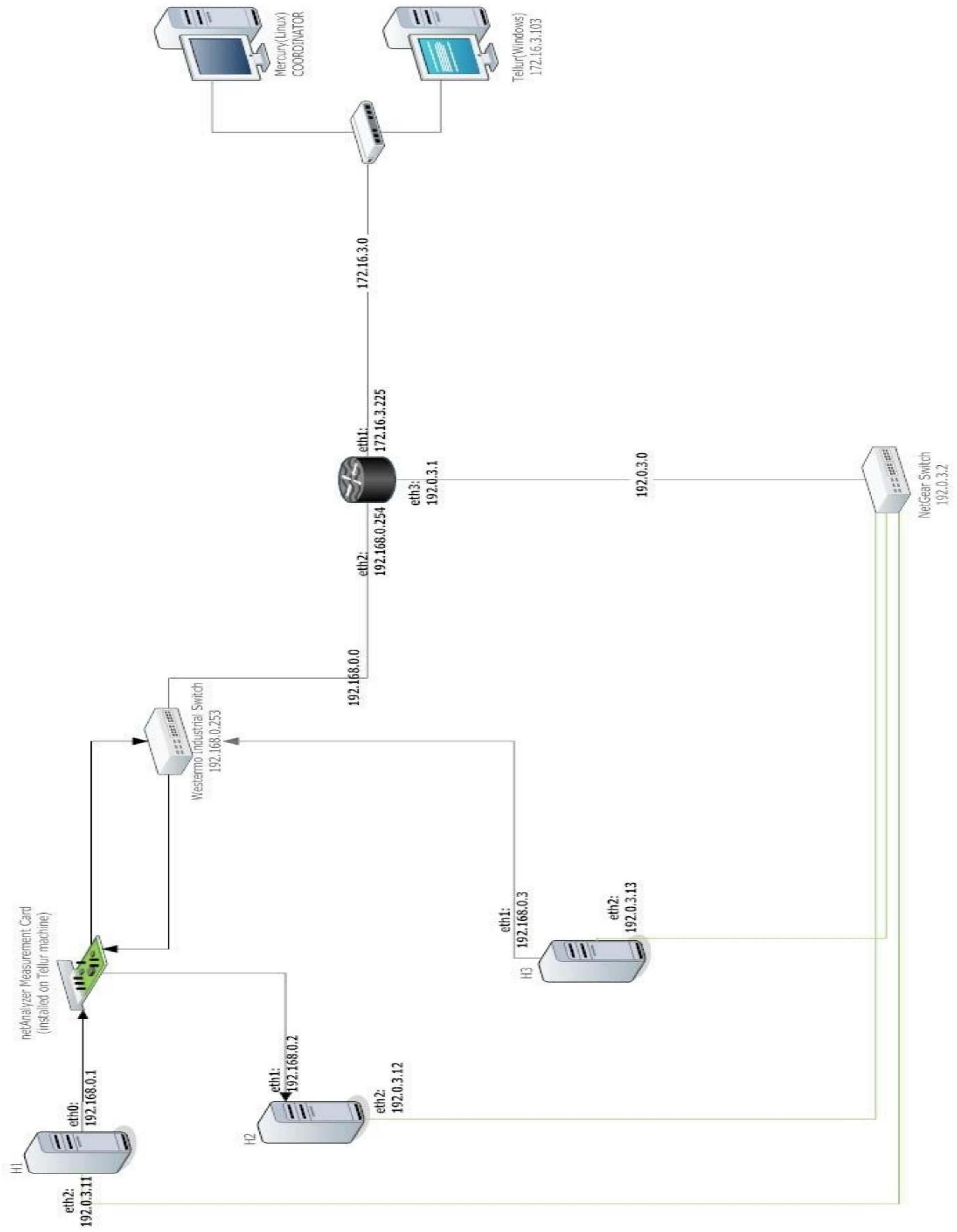


Figure 20: Network Topology, view 1

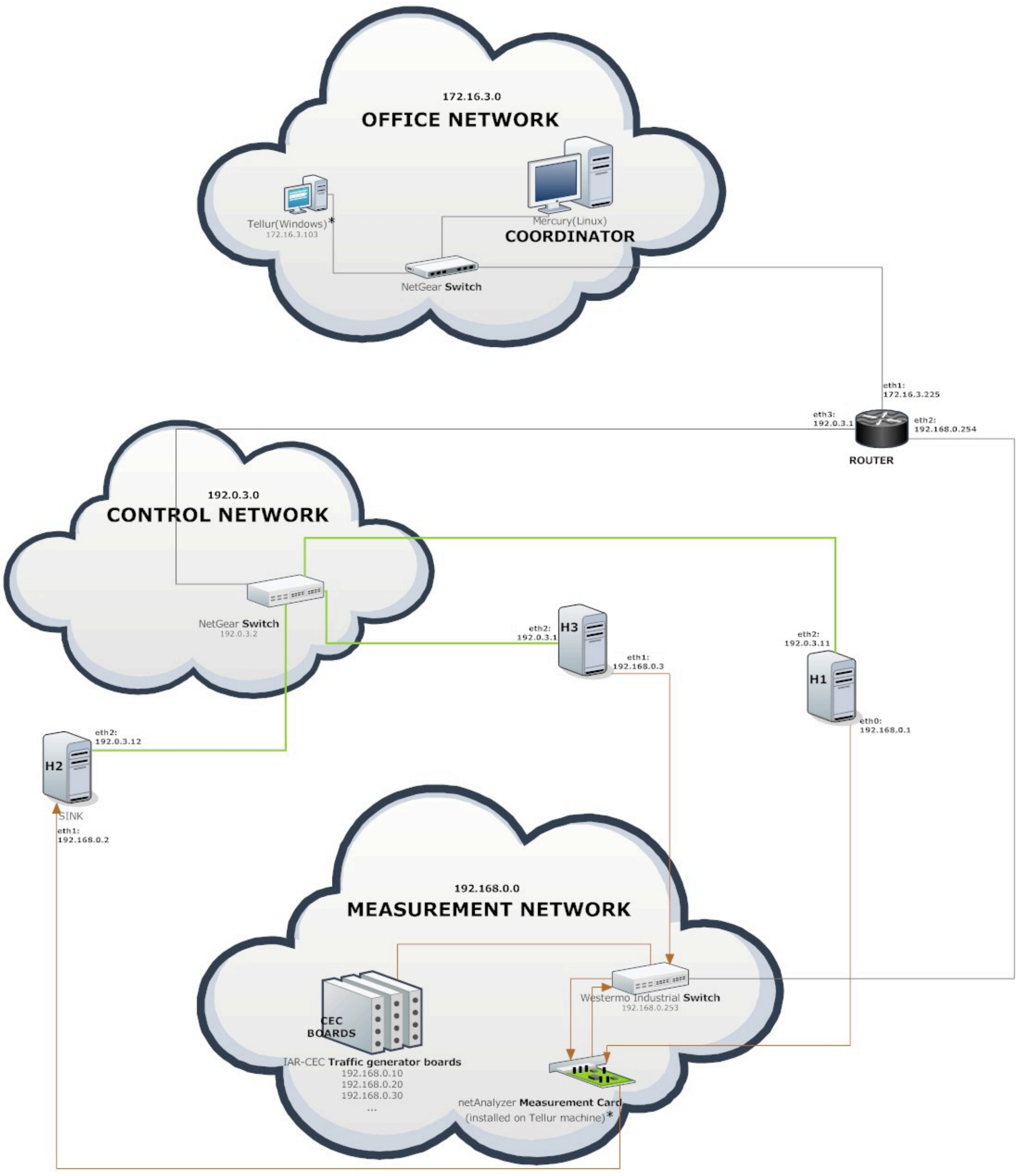


Figure 21: network topology, view 2

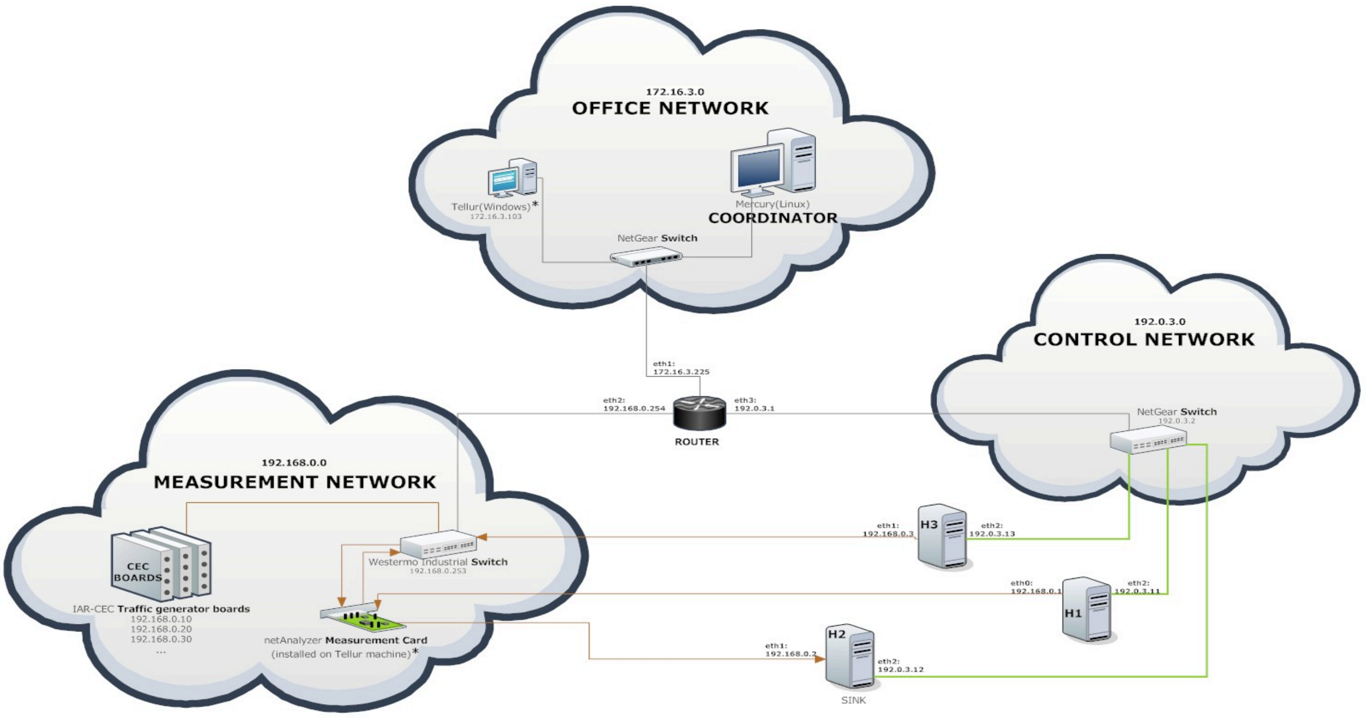


Figure 22: network toplogy, view 3

Appendix C - Coordinator Program

Next it is shown the code of the main script in the coordinator program, which calls different modules for providing the desired functionality.

```
#Define variables
capturingTime=30
((auxTime=capturingTime+6))
timestamp=$(date +"%y%m%d"_"%H%M%S")
resultDirectory=($HOME/Desktop/d$timestamp)

#add entries to local routing table
./routingTable.sh

#ping to the different hosts involved in the process
./pingFromSink.sh 192.0.3.12
(local pings not included)

#Configure the Switch
./switch.sh 192.168.0.253
sleep 2

#Create Click scripts on h1 and h3, reading the parameters from trafficConfig1/3.txt
./createScript.sh 192.0.3.11 traffic_config1.conf&
./createScript.sh 192.0.3.13 traffic_config3.conf&
wait

#Set up traffic generator boards (reads from file flow param. and send SET )
java UDPClient

#Start sink
./sink.sh 192.0.3.12 $auxTime &

#Start capturing packets in NetAnalyzer measurement card (windows machine)
./capture.sh 172.16.3.103 $timestamp $capturingTime&

# Make sure the capturing and sink programs are ready before generating the traffic
sleep 4
```

```
#Generate traffic
./trafficGenerator.sh 192.0.3.11 $auxTime &
./trafficGenerator.sh 192.0.3.13 $auxTime &

#broadcasts the START message
./broadcast.sh

#TRANSFER RESULT TO LOCAL MACHINE
mkdir $resultDirectory
./transferResults.sh 172.16.3.103 $timestamp $resultDirectory
```