Insert here your thesis' task.

Czech Technical University in Prague

Faculty of Information Technology

Department of Computer Systems

Master's thesis

Control software for the COMbo Ethernet Tester and its integration into the Netopeer configuration system

Bc. Tomáš Čejka

Supervisor: RNDr. Radek Krejčí

7th May 2012

Acknowledgements

I would like to thank my leaders from the Liberouter team. They all guide me to new experiences and knowledge. I would like to thank my supervisor RNDr. Radek Krejčí for his patience and advices. Finally and most importantly, I would like to thank my family and friends for their support, inspiration and understanding.

Declaration

In Prague on 7th May 2012 ....

Citation of this thesis

Tomáš Čejka. *Control software for the COMbo Ethernet Tester and its integration into the Netopeer configuration system: Master's thesis.* Czech Republic: Czech Technical University in Prague, Faculty of Information Technology, 2012.

Abstract

The aim of this document is a description of my work on the software part of the COMET Ethernet tester. COMET is the device based on Programmable hardware – COMBO card.

During the work on the project, I created a control software for the tester. The software includes the *libcomet* library and software utilities that use *libcomet*. The COMET software contains the command line tools, the CGI application that creates simple graphical user interface and the plug-in for Netopeer configuration system. The plug-in brings the support of the NETCONF configuration protocol for COMET.

Keywords COMET, Ethernet tester, NETCONF, COMBO card, control software, Liberouter, CESNET

Abstrakt

Obsahem tohoto dokumentu je popis mé práce na řídícím programovém vybavení nového Ethernetového testovacího zařízení COMET. COMET je zařízení postavené na programovatelném technickém vybavení – COMBO karta. Tato karta je osazena čipem FPGA.

Během práce na projektu jsem vytvořil programové vybavení testeru, které zahrnuje knihovnu *libcomet* a programy, které ji používají. Mezi vytvořené programy patří nástroje použitelné z příkazové řádky počítače, CGI aplikace tvořící jednoduché grafické uživatelské rozhraní a zásuvný modul do konfiguračního systému Netopeer. Zásuvný modul přidává do projektu COMET podporu konfiguračního protokolu NETCONF.

Klíčová slova COMET, Ethernetový tester, NETCONF, COMBO karta, řídící programové vybavení, Liberouter, CESNET

Contents

List of Listings **xvii**

Introduction **1**

1 COMET – Ethernet Tester **5**
1.1 Project requirements 6

2 Network Device Configuration and Management **9**
2.1 NETCONF 10
2.2 YANG – configuration model 13

3 Control Tools Design **15**
3.1 Hardware 15
3.2 Firmware 16
3.3 Software 16

4 Implementation **21**
4.1 PCAPEDIT 22
4.2 PCAP2SZE 23
4.3 COMSTAT 27
4.4 User interface 35
4.5 NETCONF plug-in 37

5 Testing and Verification **41**
5.1 Static Analysis of Code 42
5.2 Sender and Receiver 42
5.3 NETCONF Plug-in 45

Conclusion **49**
Future Work on COMET 49

Bibliography **51**

A Acronyms **55**

B Installation Manual **57**
B.1 Dependencies 58
B.2 Libcomet 58
B.3 COMET 58
B.4 COMET Start-up 59
B.5 Netcomet 59

C Netopeer Architecture Scheme **61**

D COMET configuration model **63**

E Contents of Enclosed CD **75**

List of Figures

2.1 The architecture of NETCONF 11
2.2 NETCONF messages exchange 12

3.1 COMBOv2 card 15
3.2 Software architecture of COMET 17
3.3 Architecture of libcomet 18

4.1 How pcapedit(1) works 23
4.2 NetCOPE Software Architecture 24
4.3 Header of frame sent from pcap2sze 25
4.4 Libcomet usage for sending traffic 25
4.5 The process of traffic sending. 27
4.6 Comstat process cycle 32
4.7 Fragments of COMET screenshots 36

5.1 Valgrind test result 41
5.2 Spirent AX/4000 control software 44
5.3 The result of the test of Packet delay histogram 46

C.1 The scheme of Netopeer 2 62

E.1 Contents of Enclosed CD 75

List of Tables

5.1 Test of packet delay histogram 45

List of Listings

4.1 ENUM format for *comstat(1)* 29
4.2 The structure cmtc_value_t 33
4.3 CNT access for *comstat(1)* 34
4.4 Change of the COMET solver stack size 34
5.1 BASH script for static code analysis (Stanse) 42
B.1 Liberouter repository configuration 57
D.1 COMET configuration data model 63

Introduction

In current days, networks are still getting faster. Meanwhile demands for quality of connections are growing too. The spread of mobile technologies causes users rely on high availability of their connection – users want to be on-line. Therefore Internet Service Providers (ISP) and network operators need to monitor, test and diagnose their high-speed network connection, in order to ensure quality services. In case of failure, they need to react flexible and promptly.

CESNET z.s.p.o. (CESNET) [11] is one of the network operators. It is the Czech National Research and Education Network operator. CESNET supplies the network connection to many Czech universities and connects the network with foreign countries.

CESNET carried out a research intent that founded the development community Liberouter [14]. The aim of the research intent[1] was the "Programmable hardware" activity. The result is a special hardware PCI card equipped with Field Programmable Gate Array chip (FPGA). It is called COMBO card. COMBO cards are possible to change their functionality in runtime and that is their advantage.

The COMBO cards bring various opportunities for hardware-accelerated network applications development. The example of the application based on COMBO card is the FlowMon system developed by Liberouter community[2].

The latest project that builds on "Programmable hardware" is a network tester. The tester is called COMET. It is an abbreviation of COMBO Ethernet Tester. COMET is a hardware accelerated Ethernet tester for optical networks. It is also based on the COMBO card. This text is about my design and implementation of the control software for COMET. The brief description of the COMET project is in Chapter 1.

The idea of a realisation of the new Ethernet tester came from the Liberouter community. There is a need to test and diagnose network devices and optical links. Such testers are usually either very expensive or mostly said they are not flexible enough. There is no Ethernet tester that can be set up from the working device in time of need. The currently used Ethernet testers

[1]Research intent MSM6383917201.

[2]FlowMon is a flow-based monitoring system based on the COMBO card. It is currently available at [3].

are physically independent devices in form of some box. That means administrators have to transport the box and connect it into the infrastructure at the place where the test will be run.

The disadvantage of these testers is the need of having them on the location. The testers are not able to do any other work than a testing and diagnosing. Our tester based on the COMBO card has this feature, because of the COMBO card. A server with the COMBO card can run any other application. When it is needed, the COMET functionality can be loaded. This tester can be run on every server with COMBO card (e.g. on every FlowMon probe) and it is loaded remotely in one minute.

Network testers usually concentrate on higher ISO/OSI layers. They test the functionality on the data link layer and the higher layers. Sometimes it is useful to check the connectivity and the link state on the physical layer. The new Ethernet tester is focused mainly on the first two layers by reading and modifying low-level registers from *Phyters*[3], which operate on the physical layer.

In order to control testers, they all have some user interface. Usually, the user interface cooperates with the devices of its vendor. However, it is difficult or impossible to use the same user interface for all of the network devices on the network. Furthermore, the control of tester is usually only locally available. Network operators and ISPs need to control network devices remotely and automatically.

The COMET tester can be controlled locally via software tools used in command line interface (CLI) on server or remotely using an SSH connection. The list of the software utilities of COMET can be found in Chapter 3 and in more detail in Chapter 4. COMET can be controlled by a graphical user interface (GUI). The COMET GUI is done by CGI application. It is designed for the Apache 2 web server [30]. However, it can be used with other servers that allow running CGI applications. The whole Section 4.4 is focused on the implementation and features of the COMET GUI.

In addition to command line tools, COMET can be integrated into the configuration system based on the NETCONF protocol. The NETCONF protocol allows users to configure and manage multiple network devices. It can be done from a central configuration and management system via secure connection. The needs of network configuration and management are discussed in Chapter 2.

Currently, the Ethernet testers can support the 40/100 Gbps speed (for example some products of Spirent Communications [29]). Our new tester works at speed of 10 Gbps, however this limitation is caused by the lack of hardware resources. The new generation of the COMBO card will bring the support of 40/100 Gbps. It is planned to be available during this year. The COMET

[3]Phyter is an independent chip on the COMBO card, that takes care of physical layer of ISO/OSI.

control software is fully compatible with the new generation of COMBO cards.

This text is divided into five main chapters. The first two chapters are general and theoretical. The COMET project is introduced in Chapter 1. Chapter 2 is about configuration and management protocols, especially about the NETCONF protocol. The most of my work on the project (the software architecture and the implementation of the control software for COMET) is placed into chapters 3 and 4. Chapter 3 describes the architecture of the whole system COMET including the software architecture. The description of an implementation of the COMET software tools can be found in Chapter 4. COMET was deployed on Liberouter development servers for testing purposes. Results of testing and verification of functionality of the COMET system can be found in Chapter 5.

Chapter 1

COMET – Ethernet Tester

The COMET project (COMbo Ethernet Tester) is an implementation of an Ethernet tester based on programmable hardware – COMBO card. The COMBO card will be briefly described in Section 3.1. COMET consists of firmware and software part. This thesis is focused on software part that I worked on.

The COMET device can be used as a transmitter and receiver of testing data. In addition, it can read information from hardware. These information are useful for the link diagnoses and tests.

The COMET device can transmit prepared/stored network traffic. It has to be hardware accelerated, because of insufficient speed of PCI Express bus, which is used for the COMBO card connection to the server. Ethernet frames that are sent to firmware can be repeatedly transmitted to the network. Repeating of the frames leads to high-speed transmission into a network and to the wanted saturation of the link. The process of transmission of data from software to firmware is described in Section 4.2.

On the other direction, COMET receives traffic and counts basic statistics. The reception of the traffic data is done mainly in the COMET firmware for the similar reason as for transmission. The COMET software obtains the basic statistics about received data. The information is used to compute some more characteristics and present them to a user. The gathering of data and the computation is described in Section 4.3.

I have created a COMET library during my work on this project. It concentrates the functionality of all control software utilities. Therefore, the source code can be easily maintained. The general description of the library belongs to Chapter 3 with the COMET architecture, specifically to Section 3.3.1. The technical details about an implementation of the library are divided into sections in Chapter 4 according to the functionalities.

As it was said in Introduction, the main advantage of this device is the possibility of operation on any current COMBO card device. In addition, it is possible to reuse some firmware modules in other project (especially the

Statistical module). It is expected, that the software module for reading hardware information could complete or replace some utilities from the Liberouter software tools.

Because of needs of network operators, the whole Ethernet tester should be controlled over a NETCONF protocol. The explanation of what NETCONF is can be found in Section 2.1. The support of the NETCONF protocol in COMET is done by integration into an existing configuration system.

The demand for the NETCONF support brings the need to create a configuration data model for COMET. During the work on the project, this model was created. It is attached in Appendix D. Analysis and design, connected with the NETCONF protocol and the configuration modelling, is described in the Section 2.

1.1 Project requirements

This section should conclude our requirements that we had at the beginning of the COMET project.

At the beginning of the project, there was a need of traffic transmission. The device should send stored network traffic at given speed. The transmission was controlled by precise timestamps of every frame, which determined the time when to send the frame to network. The COMBO card can obtain current time from a GPS module, so that it can begin transmission exactly at right time and with given interval.

The speed of transmission from software is limited by the bandwidth of PCI Express interface. Therefore, we were not able to saturate the link. In order to increase the maximal speed of transmission, hardware acceleration was required. COMET must be able to duplicate frames for given times. Software must be able to set up the hardware for this duplication.

Firmware and hardware provides a lot of information about incoming traffic and about the state of hardware. COMET must be able to read them and to present them to users. The example of data, that are necessary to be presented, is an amount of received packets, a state of link, maximal and minimal size of packet, maximal and minimal delay between packets and inter-frame gaps (IFG). The set of information, that is accessible in hardware or firmware, can differ from version to version. Therefore, COMET should have some simple way how to describe this set of information with the location where to read it.

COMET should be able to get access to the physical and data link layers and present their state to user. Hardware has various configuration and state registers and counters which have influence on the physical and data link layers. These registers are not normally (and comfortably) accessible to users. The software of COMET should be able to read these registers and allow users to change them. The example of the register is an internal loopback enable switch. When this switch is enabled, all outgoing traffic is returned to the system without leaving it. This functionality is useful during the testing of the COMBO card (there is no need to have any cable connection to test the card, when the loopback is enabled). This loopback was also used during the testing of COMET itself.

COMET should be able to work independently on any other system or device on the network. That means there has to be possibility to control it via local console. In combination with SSH protocol, COMET will be also controlled remotely. In addition, COMET should provide a simple graphical user interface in form of a simple web page, that would present read values and allow the user to change them.

The last requirement is very important. Network operators and ISPs need to be able to manage multiple devices on the network. They have a lot of other requirements on the configuration and management system. The NETCONF protocol is designed to fulfil this task. COMET must support the NETCONF protocol. It should be done by integration of COMET as a plug-in into some existing NETCONF system. It was chosen the Netopeer configuration system, which is developed by CESNET. This system implements the NETCONF protocol and provides a NETCONF library. The Netopeer system is mentioned in Section 2.1.1.

Chapter 2

Network Device Configuration and Management

Large networks are usually hard to maintain, especially if there are devices from different manufacturers on the same network. For these heterogeneous networks, there is a lack of unified configuration systems. It would be very time-consuming and sometimes impossible to configure every device of the network separately.

That is the reason why network administrators and operators need some centralized and standardized configuration and management system. To solve this problem, some configuration and management protocols were developed. The most known is probably SNMP (Simple Network Management Protocol) [10, RFC2570]. SNMP can be used for obtaining data from devices and originally also for setting new values of configuration registers.

The disadvantage of SNMP is the lack of complexity of configuration possibilities that current devices support. Modern network devices usually allow users to configure a lot of things but not via SNMP. Generally, manufacturers pay more attention to their own proprietary user interfaces to present configuration to users than to standard configuration protocols such as SNMP. SNMP seemed to be difficult too much for configuration purposes, because of concentration on data-oriented abstraction level. Network operators desire task-oriented abstraction level. As the result, using just SNMP is generally insufficient in comparison to CLI of devices.

Some of the advantages and disadvantages of SNMP and other ways of device configuration brings up the [27, RFC3535]. This RFC is not meant to be a complete list of features or requirements that network operators expect. However, it is the first movement to the solution of the problems with device configuration.

To solve problematic situation with complicated way of multiple network device configuration, people around IETF (Internet Engineering Task Force)[4] decided to develop a new configuration protocol called NETCONF.

2.1 NETCONF

The NETCONF protocol was prepared by IETF with cooperation with network operators and commercial companies. The name NETCONF stands for "Network configuration". It is a configuration protocol described in RFC documents. The first document is [16, RFC4741] (proposed in December 2006), where we can find general introduction. This document was later revisited by [17, RFC6241] (proposed in June 2011).

The main resources of information about the NETCONF protocol are administrated by NETCONF WG (working group). In addition to materials available on NETCONF WG server, there is also the server [5, netconfcentral.org], which concentrates materials and tools connected with NETCONF and YANG (Section 2.2).

The motivation of NETCONF WG was to create a protocol that "operators want to use" and "vendors want to implement". Network operators need to manage networks, not just network devices. Vendors usually want to implement simple and efficient protocols.

For automatic configuration of multiple network devices, network operators used various scripts that interacted with CLI of devices. Devices were configured by these scripts. The NETCONF protocol tends to substitute this way of configuration. It supplies the way of native automatic configuration of the whole network.

Architecture of the NETCONF protocol is shown in Figure 2.1[5]. It can be partitioned into four layers. The layers are: Secure Transport, Messages, Operations and Content.

On the bottom of the architecture model, there is a Secure Transport protocol. From the Figure 2.1, it is obvious that the NETCONF protocol has no strict dependency on a transport protocol. There are drafts about NETCONF using SSH, TLS, BEEP/TLS or SOAP/HTTP/TLS, however the SSH protocol is mandatory.

The second layer from the bottom is the Messages layer. A NETCONF message can be an RPC request, reply or a notification (`<rpc>`, `<rpc-reply>` and `<notification>`). These messages are well-formed XML documents.

[4]Definition found on www.ietf.org: "The Internet Engineering Task Force (IETF) is an organized activity of the Internet Society (ISOC). ISOC is a non-profit organization founded in 1992 to provide leadership in Internet-related standards, education, and policy. It is dedicated to ensuring the open development, evolution and use of the Internet for the benefit of people throughout the world. See: "www.internetsociety.org".

[5]The figure was originally published in [17, RFC 6241] as an ASCII graph. For the purpose of this text, it was repainted using *dia(1)*.

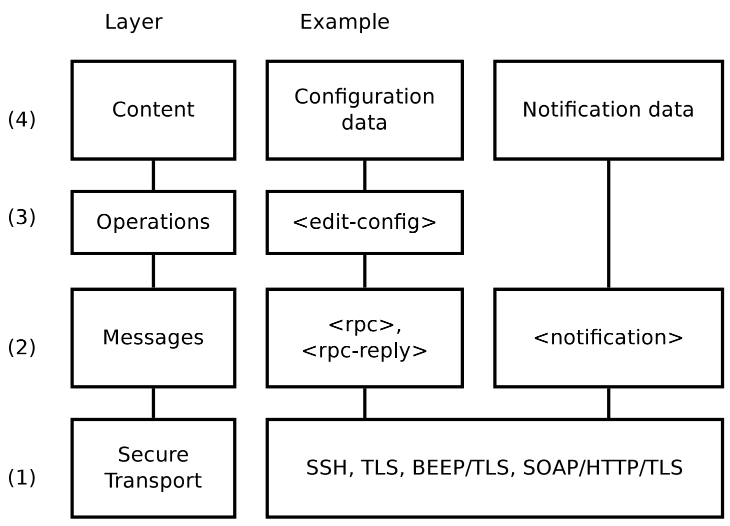


Figure 2.1: The architecture of the NETCONF protocol is divided into four layers.

The third and the forth layers contain an operation and configuration data. The operations defined by RFC are:

<get-config> — retrieves the configuration information,

<get> — retrieves the configuration and state information,

<edit-config> — modifies the data in configuration datastore,

<copy-config> — copies the datastore,

<delete-config> — deletes the datastore except *running*,

<lock> — lock entire datastore,

<unlock> — unlock datastore,

<close-session> — graceful termination of a NETCONF session,

<kill-session> — force termination of a NETCONF session.

The set of mandatory operations is supported by the COMET.

NETCONF is a client-server protocol. The NETCONF server is running on a target device and it is listening for incoming connections. After successful connection of a client over SSH, the client and the server begin to exchange NETCONF messages. Exchange of messages shows Figure 2.2. The NETCONF protocol defines own session that must be established. The establishment of new session begins with *hello* message. *Hello* message contains

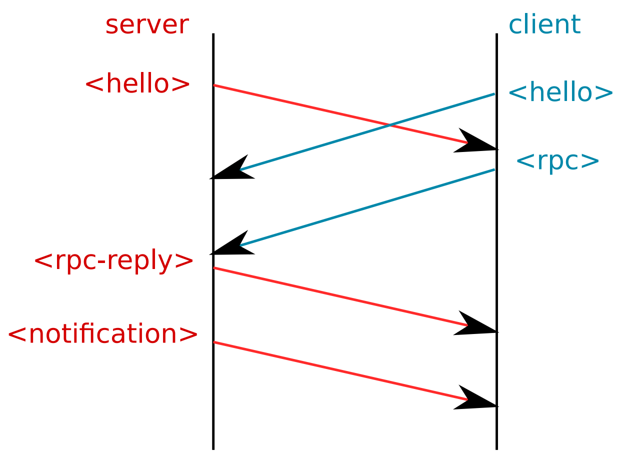


Figure 2.2: NETCONF messages exchange[6]

supported features, so called *capabilities*, that client and server can handle. The following communication uses the common subset of capabilities.

The NETCONF protocol defines configuration datastores for *candidate*, *running* and *startup* configuration. Administrators can prepare candidate configuration and apply it when everything is ready. NETCONF also supports concurrent work by operations *lock* and *unlock* and transaction with rollback. The *candidate* datastore is the example of capabilities defined in RFC.

According to performance tests, the NETCONF protocol is suitable for a configuration of large networks with many devices. It is more efficient than SNMP in bandwidth consumption. Results of the tests were presented in the article [21] about the protocol efficiencies of NETCONF versus SNMP.

2.1.1 Netopeer

Netopeer [24] is an open-source implementation of the NETCONF protocol written in the C language. It was originally created as a bachelor's thesis [22] at Masaryk university in 2007. The further development is run by the CESNET organization. That is the reason the Netopeer system was chosen as a NETCONF system for COMET. The second version of Netopeer – Netopeer 2 is currently under development. In this text, Netopeer 2 is referred as Netopeer.

Netopeer uses the NETCONF library called *libnetconf*. *Libnetconf* was a part of the Netopeer project; however it is currently a separated project. Source codes of *libnetconf* are available on the project web sites [23]. *Libnetconf* is intended for building NETCONF clients and servers. It provides functions to establish a NETCONF connection between client and server via the SSH protocol. Furthermore, it is used to send and receive NETCONF messages and to work with stored configuration data.

The complete architecture of Netopeer is published as a scheme available on the official project websites. The copy of the scheme is attached in Appendix C, Figure C.1. The most important part of Netopeer system is a NETCONF server – *netopeer-server*. The COMET plug-in is loaded as a "Device config module" by this server. Netopeer system passes the request to "Device config modules" and waits for their reply.

Besides the NETCONF server, there is a NETCONF agent (*netopeer-agent*) on the server part of the system. *Netopeer-agent* is an SSH Subsystem, that is executed when an SSH connection for the NETCONF Subsystem occurs. *Netopeer-server* accepts requests from a client via *netopeer-agent*. The requests are passed to plug-ins to process them. The plug-in is a dynamically linkable file. The generated response is sent back to the client.

The COMET plug-in for Netopeer system can be found in the *netcomet* directory on enclosed CD (Appendix E). COMET uses *make(1)* and Makefile to compile the dynamically linkable plug-in for Netopeer. More information about the COMET plug-in can be found in Section 4.5.

2.2 YANG – configuration model

A configuration and management protocol must be fully decoupled from the configuration data model. Therefore, new working group NETMOD separated from NETCONF WG. NETMOD WG develops the YANG modeling language. YANG is a modeling language used for a device configuration description. It is described by series of RFC documents, where the main one is [9, RFC6020].

According to original idea, YANG is "built on existing MIB[7] and SMI[8] experience". SMI and MIB were created as a supplement for SNMP. It is a virtual database of configuration and status entities. YANG was designed to be very extensible and authors planned the usage of SMI and MIB modules created in the past.

The YANG language is platform and device independent. The idea of a YANG model is to create an interface definition between client and server. It is a little bit similar to the CORBA [2] philosophy, where a special language called IDL is used to create a skeleton and a stub of application. YANG model should be enough for client/user to know what the incoming data from the response mean. The model should contain a textual description of elements.

The Syntax of the YANG language is quite simple and a bit similar to the C programming language. There are defined keywords such as *container*, *leaf* and *list*. Each of the keyword has its own semantic and a set of elements that can follow. According to RFC:

> YANG module contains a sequence of statements. Each statement starts with a keyword, followed by zero or one argument,

[7] [4, Management Information Base]
[8] [6, Structure of Management Information]

followed either by a semicolon (";") or a block of substatements enclosed within braces ("{ }"): statement = keyword [argument] (";" / "{" *statement "}")

YANG model has a tree structure so that it can be converted to an XML document. The conversion is useful for easier processing with existing tools and libraries. As RFC says, YANG modules can be translated into an equal XML syntax called YANG Independent Notation (YIN). For example the program *pyang* [1] can be used to convert models between YANG and YIN formats. *Pyang* is written in python and it is able to generate schemas for validation and it can be used as a data validator.

Examples of YANG models could be found either in RFC documents or on the server netconfcentral.org [5]. During the work on COMET, a configuration model was designed. The first draft of the COMET configuration model was designed by Ing. Ladislav Lhotka, CSc. It was done in the YIN format.

To make the COMET model compatible with the Netopeer system, I have made some modifications of the model. At first, it was needed to “concatenate” or merge the model into one XML tree. From this tree, I removed some unused *leaves*, such as “Reset after Read” bits, that were not useful for end users. On the other hand, it was necessary to insert some new RPC operations (e.g. *send-traffic* operation). The resulting configuration model in YANG is in Appendix D.

The COMET tester model consists of at least one interface (container in YANG terminology) called *comet-tester* in the model. The *comet-tester* container is divided into the COMET parts:

statistics — contains information from Statistical module,

sender — module for sending a network traffic,

output-packet-checker — enables special tag insertion during transmission,

input-packet-checker — check of incoming tagged traffic,

blackhole — frame dropping for incoming traffic (to keep input queues free and prevent discarding),

physical-coding-sublayer — contains low-level information from hardware.

The rest of the model is the list of RPC operation definitions:

reset-modules — reset modules to initial state,

reset-counter — set counters to zero,

send-traffic — start of transmission.

Chapter 3

Control Tools Design

The architecture of the COMET project is shown in Figure 3.2. This section is focused on the layers of architecture. The architecture can be divided into three parts – hardware, firmware and software. The chapter begins with the hardware layer and goes up to the software. The COMET software is my part of project, so it is described in detail.

3.1 Hardware

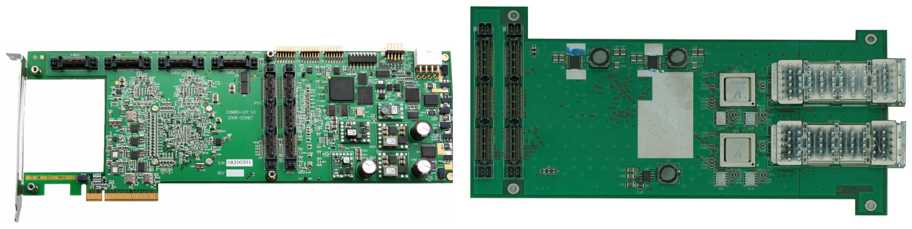

(a) COMBO-LXT (b) COMBOI-10G2

Figure 3.1: COMBOv2 card

At the lowest layer of the architecture of COMET, there is a COMBO hardware card. COMBO card consists of COMBO-LXT Express PCIx8 mother card (Figure 3.1(a)) and some of the interface cards. COMBO-LXT belongs to the second generation of COMBO cards (COMBOv2) and it is the currently used model. COMBO-LXT card is equipped with the FPGA chip XILINX Virtex 5[9].

[9]XC5VLX110T or some other compatible chip, the technical detail specification can be found on Liberouter web pages [12].

> Even though the generations of the cards differ, for the purposes of this text, COMBO and COMBOv2 names are interchangeable. The first generation of the card is not supported any more. Therefore, COMBO card means COMBOv2 in this text.

The FPGA chip is the main processing unit of the COMBOv2 card. It must be loaded with special firmware. The firmware determines behaviour and functionality of the device. The firmware of the COMET project is described in Section 3.2.

The connection of the COMBO-LXT card into the optical network is done by a COMBO interface card. As an interface card, it can be used COMBOI-10G2 (Figure 3.1(b)) with two XFP cages. This interface is currently used on the Liberouter development servers. Therefore, the COMET project primary supports this interface card.

3.2 Firmware

To start the Ethernet tester COMET, a special firmware must be loaded into the FPGA chip. The COMET firmware was designed and implemented by Bc. Pavel Benáček during his work on a master's thesis [8]. The result of his work was a design file, that is located on the CD (Appendix E) in the */fw* directory. The firmware is loaded by executing a COMET initialization script (see Installation Manual in Appendix B).

The COMET firmware is built on the NetCOPE platform [13][10]. NetCOPE contains building blocks for rapid development on the COMBO card. The COMET firmware adds its own functional blocks such as Histogram module.

Histogram module is used for counting of packet amounts. Histogram module contains bins of configured ranges. Every bin is used as a counter of packets with the same characteristic. The examples of Histogram module are the histogram of packet sizes and histogram of inter-frame gaps in Figure 4.7. The advantage of Histogram module is the possibility to obtain statistical information about the network traffic, however there is no need to transfer all the traffic data.

Furthermore, there are other COMET firmware modules. The complete list can be found in [8].

3.3 Software

Figure 3.2 represents the logical structure of the COMET system. The lowest layer was described in Sections 3.1 and 3.2. This section is focused on the

[10]NetCOPE Software architecture is discussed in Section 4.2, because of the communication with hardware.

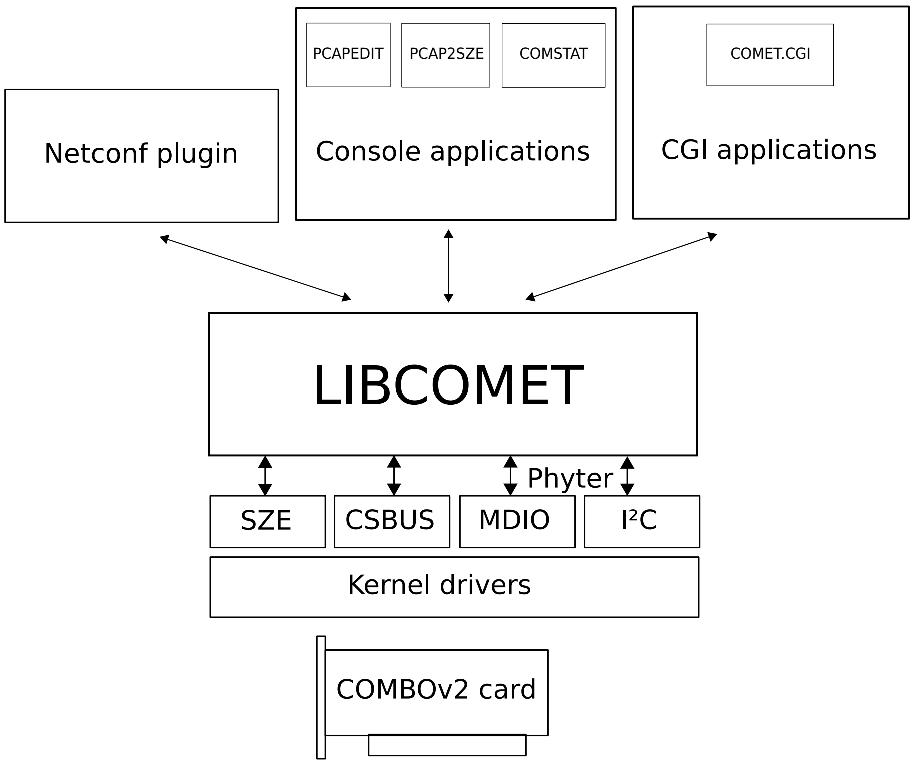


Figure 3.2: Software architecture of COMET — The software architecture is based on kernel drivers and Liberouter libraries. The COMET functionality is implemented in the *libcomet* library. *Libcomet* is used in all COMET applications.

software layer that begins with kernel drivers.

Above the firmware, there are Kernel modules (drivers) for COMBO card. Kernel module detects the COMBO card as a PCI Express device. The COMBO card has its own identification number, known to Kernel driver. The identification number determines the firmware design that is loaded into the FPGA chip. In order to make COMET design supported in COMBO drivers, I had to modify the Kernel module.

Kernel modules make the hardware card accessible to software. The communication with Kernel modules is done via *libsze2* and *libcombo* Liberouter libraries. These libraries carry out SZE, CSBUS, MDIO (Management Data Input/Output) and I^2C communication interfaces, shown in Figure 3.2. The explanation of the used communication interfaces can be found in Section 4.3.1.

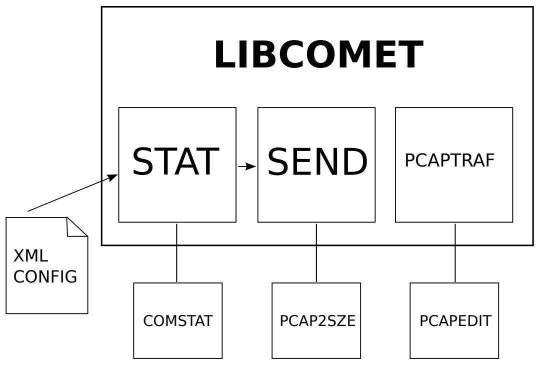


Figure 3.3: Architecture of libcomet

3.3.1 Libcomet

The COMET software, that I implemented, can be found on the first half from top of Figure 3.2. In the lower row of the half, there is the *libcomet* library. *Libcomet* contains the main functionality of the COMET tester.

Libcomet consists of three basic modules. The structure of *libcomet* is shown in Figure 3.3. This section will go through the *libcomet* modules.

The first module (**PCAPTRAF**) can be used for a frame manipulation in a PCAP file. Currently, this module supports the basic set of operations. These operations are listed and described in Section 4.1.

The second module (**SEND**) can send a content of given PCAP file according to timestamps. This module fulfils the requirements on the full speed transmission and the transmission in given intervals. It is used to reply network traffic dump in PCAP format. That means that content of the input file is resent from the COMBO card to the network. This module is described in Section 4.2.

The third module (**STAT**) communicates with hardware and obtains information about hardware state and an incoming traffic. It can be used to change configuration registers, disable the transmitting and receiving chips, enable internal loopbacks and so on. The list of accessible registers and values can be found in the special configuration file on enclosed CD (Appendix E) in the

```
/comet/sw/comstat
```

directory. The module, its functionality and the configuration file are described in Section 4.3.

3.3.2 Software Tools

There are three kinds of the software tools – command line tools (console applications), CGI application and Netopeer plug-in. All of these tools use the *libcomet* library. Console applications correspond to *libcomet* modules that were mentioned in Section 3.3.1. Description of the console applications can be found in Chapter 4. That chapter contains an explanation of how the *libcomet* modules work.

The names of the library modules represent the header files of the *libcomet* library. That means *libcomet* has *stat.h*, *send.h* and *pcaptraf.h* header files. The main library header file, that contains an inclusion of all three files, is called *comet.h*. After an installation of the library, all header files are located in the *libcomet* subdirectory.

Figure 3.3 shows an input file of the library. It is the "XML config" box in the figure. This file is called *addresses.xml*. The meaning and the structure of the "XML config" file is explained in Section 4.3.2.

There is a connection between STAT and SEND modules. The connection is shown in Figure 3.3 as an arrow. It means that STAT provides some functionality for the SEND module. The SEND module searches for the address of a register with the current time.

The second kind of the software tools, which was mentioned in the beginning of this section, is a CGI application. CGI application of COMET is called *comet.cgi*. It is a simple graphical user interface of COMET. It is based on the STAT module that obtains information from hardware. The graphical user interface has the similar functionality as one of the command line tools (*comstat(1)*). However, the usage is more comfortable. Section 4.4 is focused on the graphical user interface of COMET.

COMET supports the NETCONF protocol (explained in Section 2.1). It is done by an implementation of a plug-in for the Netopeer system. The Netopeer plug-in of COMET is located in the *netcomet* directory. Therefore, the plug-in is also called *netcomet*. It uses the STAT and SEND modules of the *libcomet* library. Section 4.5 is focused on *netcomet*.

Chapter 4

Implementation

This chapter is divided into sections according to the *libcomet* modules. Every section has the same name as the corresponding console application that uses the library module.

This chapter is focused on technical details about software utilities included in the COMET project. The COMET project was developed in CESNET environment, using CESNET's network and servers. The source codes of the most of software utilities are written in C programming language. Project also includes some python scripts (such as plug-in for *pyang*) and during the development, I created some bash scripts (e.g. initialization scripts).

The source codes of the COMET software including all scripts, COMET utilities and COMET library, are placed on the enclosed CD. The source codes can be found in the

```
/comet/sw/
```

directory. Everything can be freely shared under the BSD licence.

COMET was developed and continuously tested on the Liberouter development servers with COMBOv2 cards. These servers were located mainly in Brno (the Czech Republic). The most of my work was done remotely via SSH connection.

Compilation of the whole project is done using the GNU build system known as Autotools. It is a suite of tools for build automation (GNU Autoconf [18], GNU Automake [19], GNU Make [20]). The Autotools system allows recompiling of the project just by typing:

```
 ./configure && make
```

in the root directory of the project[11].

[11] *Libcomet* and *netcomet* are compiled separately from the software utilities.

A documentation of the project is mainly generated from the commented source codes by the Doxygen documentation system [15]. Doxygen can be configured to generate documentation in various formats. For *libcomet*, there is HTML, LaTeX and man documentation. For software tools, it is set to generate only HTML version of manual. Man pages for tools are written separately from Doxygen. The documentation can be found on enclosed CD.

As it was mentioned in Chapter 1, there are two main parts of COMET – one for transmission and one for receiving. Receiving of the network traffic is done only on hardware layer. COMET has a special module in the firmware that gathers statistics about an incoming traffic. The statistics are read, counted and displayed by *comstat(1)* (Section 4.3).

The transmission is done via application called *pcap2sze(1)*. This tool loads the list of frames from a PCAP file and sends data into the COMBO card. Section 4.2 is focused on *pcap2sze(1)*.

4.1 PCAPEDIT

Pcapedit(1) is a software utility for manipulation with PCAP files. I developed this utility to change the content of a PCAP file. The program reads a given file and iterates over stored frames.

The PCAP file format is used for network traffic storage. To implement the reading of PCAP file, I needed to know its structure. PCAP files contain headers of two types. The first header is global for the whole file. The first header contains the size of the longest stored frame. According to this value, enough memory for reading buffer can be allocated. The second type of header is added to every frame. It contains information about timestamp, the frame length and the length of stored part of the frame. The timestamp of a frame is used in *pcap2sze(1)*.

A function for PCAP file reading was needed in *pcap2sze(1)* to obtain data for sending. Therefore, it was placed into the COMET library *libcomet* as a function called *cmte_load_file()*. It is used in both *pcapedit(1)* and *pcap2sze(1)*. *Libcomet* creates a list of frames using circular queue from *queue(3)*.

The *pcapedit(1)* program loads PCAP file and then perform operations on the set of stored frames. The available operation set includes:

load file — load contents of file,

show frames — print a list of frames,

clone frame — make a copy of frame,

remove frame — remove a frame,

remove frame interval — remove a list of frames given by the range of indexes,

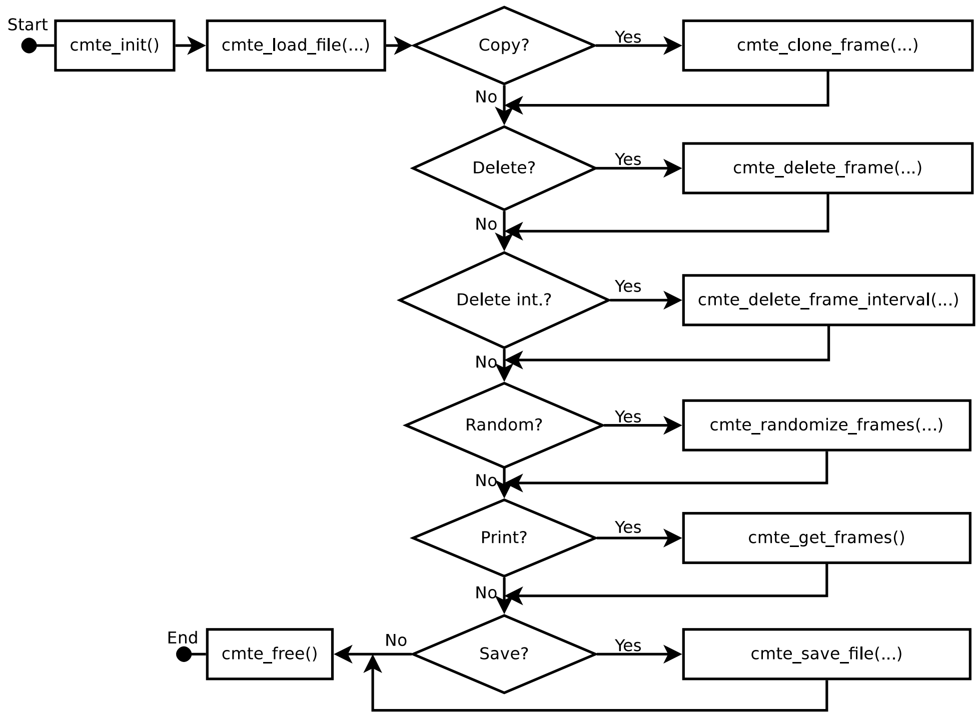


Figure 4.1: How pcapedit(1) works

save file — save content of internal frame list into the file,

randomize frames — this function was developed for anonymisation of addresses in PCAP files and it was used to process data in the bachelor's thesis of Michal Michalík [26].

The executed *pcapedit(1)* utility behaves according to Figure 4.1. The figure contains the names of library functions. The names of the functions are obvious and correspond to the *pcapedit(1)* operations listed in this section.

From the figure it is clear that *pcapedit(1)* has the fixed order of execution of operations. However, it was sufficient for our manipulation with PCAP files. Having the main functionality implemented in *libcomet*, the behaviour of *pcapedit(1)* (e.g. the order of operations) can be easily changed by source code modification and recompilation. The parameters of *pcapedit(1)* can be found in manual page.

4.2 PCAP2SZE

The program for sending traffic to a network is called *pcap2sze(1)*. The name is derived from the names PCAP and SZE. Section 4.1 explained the meaning of PCAP. A PCAP file is an input for *pcap2sze(1)*.

SZE is a communication interface of COMBO drivers, which were developed in Liberouter team's laboratory. In Figure 4.2 it is the lowest layer. The COMBO drivers can be divided into two logical parts. COMET uses both communication interfaces via Liberouter libraries *libsze2* and *libcombo*.

In the introduction of Section 3.3, there is a mention about the COMBO drivers modification. The COMET firmware has its own identification number. *Szedata2* checks the identification number in order to decide whether the hardware supports the SZE channel. In order to get drivers know the COMET firmware identification, the COMET identification number had to be added to the COMBO drivers[12].

The "combo" part of drivers supplies the interface for small volumes of data transfer; "szedata2" provides an SZE interface. The SZE interface is used for a fast transfer of large data. Using ring buffers in SZE, data can be passed on between the card memory and the host memory.

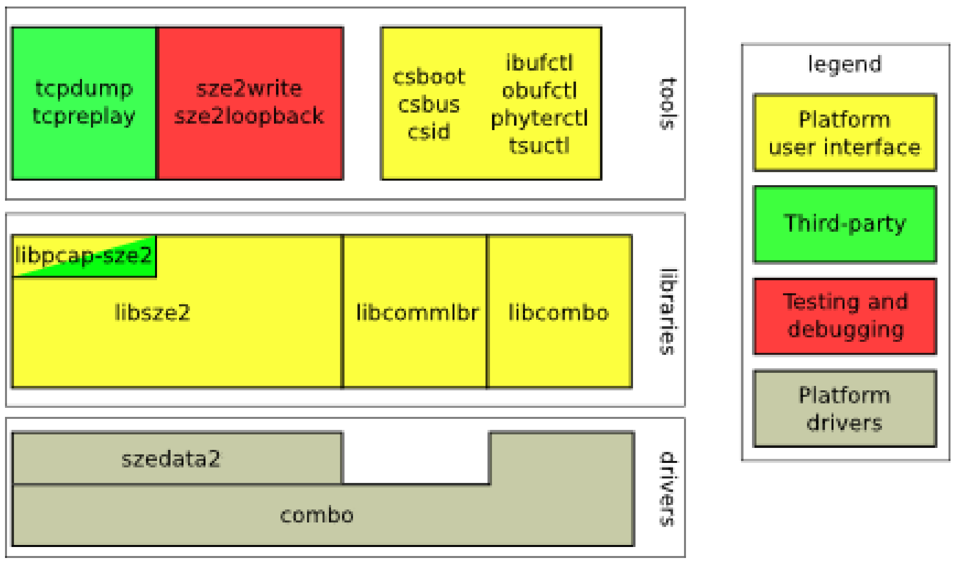


Figure 4.2: NetCOPE Software Architecture – This figure was taken from the NetCOPE Platform Handbook, written by Liberouter team.

The loading of PCAP file was described in Section 4.1. *Pcap2sze(1)* sends all frames from the circular queue into the COMBO card via the SZE interface. For communication via SZE interface there is a library called *libsze2*. *libsze2* implements a function used for sending data. *Pcap2sze(1)* uses this function to send data (the content of PCAP file without headers) with special a header (it is shown in Figure 4.3).

[12]The identification number for COMET firmware was chosen as 0*xc*0330100 for its similarity to "COME0100" after the rotation of numbers. "0100" means the first version of COMET.

The header is processed by firmware and it is used for an assignment of the timestamp and other settings for the frame. The structure of the header is declared in *pcapfile.h* in *libcomet*. The whole header has 13 B and it is declared with attribute *__packed__*, which tells to the compiler not to align the structure size.

Nowadays, only one least significant bit is used from the "flags" byte. This bit selects a mode of the timestamp interpretation. There are two possible modes – relative (when value is '1') and absolute. The last entry of the header is "repeat". It means the number of repetition of current frame.

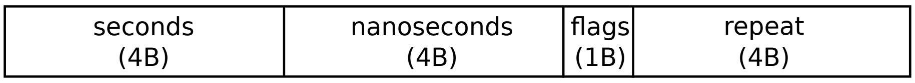


Figure 4.3: Header of frame sent from pcap2sze

Frames are queued by the COMET firmware with its headers. They are resent to a network one by one exactly when the time is greater or equal to timestamp from the header. For relative mode, firmware can wait for the interval that is set by timestamp, before the next frame is sent. In the relative mode, the COMET firmware expects the time in bytes instead of seconds and nanoseconds. The conversion is done automatically by software. In order to get exact time, there is a GPS module connected into the system.

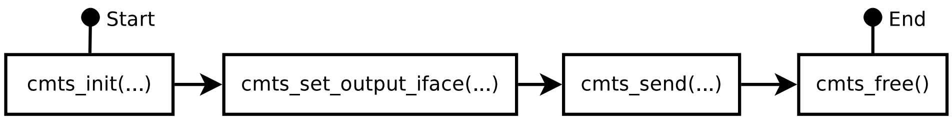


Figure 4.4: Libcomet usage for sending traffic

The usage of the *libcomet* module for sending is shown in Figure 4.4. At first, *libcomet* needs to be initialized by the *cmts_init()* function. After that, user can specify the output network interface with *cmts_set_output_iface()*. The function *cmts_send()* is the most important – it sends the content of given PCAP file according to given parameters. Information about the amount of sent frames is available in the variable:

```
uint64_t cmts_packet_sent
```

The function *cmts_free()* is a destructor of the *libcomet* module and it is used to free resources of the module.

The function *cmts_send()* is described in more detail in Figure 4.5. The most of the source code of *cmts_send()* is concentrated in a loop. The loop iterates over the list of frames in circular queue (after loading the content of PCAP file).

In every iteration, the function determines the timestamp of the current frame. There are three modes of sending:

- **User interval** — the interval between two following frames is fixed,
- **User speed** — user specified the speed of transmission (in bytes per second),
- **Time from PCAP** — the intervals are taken from the loaded PCAP file.

After computing a timestamp of the current frame, there is an "Apply move" operation. It adds a constant time in case of delayed start of transmission. The time of delayed start can be given as a parameter of the function. When there is no delayed start time specified, the timestamp is not changed.

"Normalize time" operation converts the resulting timestamps to the valid time. That means if the amount of nanoseconds is greater than the constant `NSECS_IN_SEC` (amount of nanoseconds in one second), nanoseconds are divided and seconds incremented.

```
seconds = nanoseconds / NSECS_IN_SEC
nanoseconds = nanoseconds % NSECS_IN_SEC
```

At the end of every iteration, there is a test whether to send the frame to a network. If the "do not send" flag is set, the frame is not sent. The computed timestamp can be displayed in verbose mode even if the frame is not sent.

The *pcap2sze(1)* program has a set of parameters. All parameters can be found in manual page. Users can modify the first timestamp (`-t`) or they can use the timestamp from the PCAP file. The beginning of the transmission can be delayed by "Applying move" (`--move`). The intervals between frames can be set manually (`-i`) or they are computed according to given transmission speed or they are taken from the intervals between frames in the PCAP file.

The interesting parameter of *pcap2sze(1)* is `-g`. It causes the program obtains the current time from the COMBO card and uses it as a timestamp of the first frame. Reading the time is done by the module of *libcomet* described in Section 4.3.

It is clear from previous paragraphs that the COMET software, especially the *pcap2sze(1)* program, needs to use functions for timestamp computation. Some of these functions originally were intended to be implemented in *pcap2sze(1)*. They were moved to Liberouter's *libcommlbr* library[13], so that they can be used in other projects. These functions can be found as "Number manipulating functions" and "Time functions" except the *cl_timeval_diff()* function that was already written.

[13] *libcommlbr* is the Liberouter Common library, it contains functions that are used in various Liberouter projects.

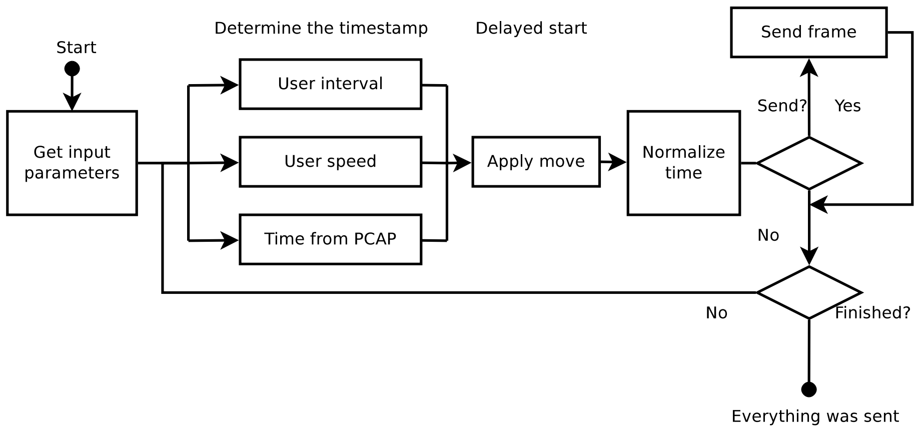


Figure 4.5: The process of traffic sending.

4.3 COMSTAT

The *comstat(1)* program was intended for reading information only for the COMET project. Because it is quite universally designed, its possibilities are highly extensible and flexible. The whole program uses one configuration file:

This file contains the list of “items”, that *comstat(1)* can work with. It means *comstat(1)* can read a value of the “item” from hardware. The value of some “items” can be changed (for Read-Write “item”). The structure of the configuration file will be described in Section 4.3.2.

As a result, *comstat(1)* can be used for obtaining information, not necessary from COMET. In addition, in case of changes in the firmware or hardware we do not need to recompile *comtat(1)*. A modification of the “item” set is done just by simple editing configuration file.

The COMET firmware contains a lot of counters. The way how they are read is discussed in Section 4.3.1. The most of the counters are mainly 64-bits long in order to prevent overflow. They are concentrated in the Statistical module.

The Statistical module is the most important for COMET, because it contains information about incoming traffic. To show the characteristic of the flow, the Statistical module contains some histogram units. Currently, we have histograms of packet sizes, packet delays and inter-frame gaps. The histogram unit is described in more details in the master’s thesis [8].

4.3.1 Communication with hardware

Reading of all incoming raw data from hardware to software is not possible at full speed (the bandwidth of PCI Express is seriously limited). That is the reason why the hardware does the most of the computation and software mainly reads the values from counters.

According to the division in Figure 4.2, the COMBO card has two main driver modules. The SZE (*szedata2*) module was mentioned in Section 4.2 as one of the possible way of communication with hardware. *Comstat(1)* uses rather the *combo* module interface than SZE. In the COMBO card firmware, *combo* communication interface is based on the MI32 protocol. This protocol defines the ways of communication between modules. MI32 endpoint is a firmware module that is accessible (addressed) from the software. To make the registers and the counters accessible, the author of the COMET firmware created MI32 endpoints.

A communication with an MI32 endpoint can be tested by the *csbus(1)* tool. It is one of the Liberouter software tools. That is why this interface will be called CSBUS in this text.

The process of register reading via CSBUS is performed in two steps. The first step is to map a memory that will be available from the userspace. Then, it is possible for drivers to copy the content of the target memory. The used functions are *cs_space_map()* and *cs_space_read_4()* from the *libcombo* library.

The histogram values of the COMET histograms are read specifically. The histogram module is always controlled by one configuration register. Values of all bins from one histogram are read from one value register repeatedly. The COMET histogram has something like an inner pointer that selects the current bin to read the value from. This pointer can be reset by setting one configuration bit of the configuration register. After one reading from the value register, the pointer is incremented automatically, so it points to the next value bin.

The current version of *comstat(1)* supports two communication interfaces (buses) for reading data from hardware. These interfaces are implemented in the *libcombo* library. Statistical and Packet Checker modules are developed to communicate via CSBUS, most of the phyter registers are available with MDIO interface. It is planned to add the I^2C interface too, because it is used for a lot of low-level registers from phyters.

MDIO is a serial bus. It is used for the Media Independent Interface (MII) that connects Media Access Control (MAC) devices with Ethernet physical layer (PHY). MDIO is connected for example to Physical Coding Sublayer (PCS) and Physical Medium Attachment sublayer (PMA). This way PCS and PMA registers can be read with *comstat(1)*.

4.3.2 Configuration file

During the project development, I created two versions of configuration files for *comstat(1)*. The latest version of the configuration file is called:

```
addresses.xml
```

This file is a well-formed XML document. The structure of the XML tree is historically built from the previous version. The previous version was a plain text file with data organized in rows. The new version that uses the XML technology is more flexible. Therefore, the plain text file is no more supported.

This section is focused on details about the `addresses.xml` file structure and configuration possibilities. Listing 4.1 will be used for description of meaning of elements of configuration structure.

```xml
<?xml version="1.0" encoding="utf-8"?>
<device>
 <modules>
  <module bus="CSBUS" header="Control modules">
   <addresses><address ba="0xC00000"/></addresses>
   <items>
    <item>
      <name>CM_DEVNUL</name>
      <offset>0000</offset>
      <bitmask>0x1</bitmask>
      <size>W</size>
      <access>RW</access>
      <format>ENUM</format>
      <display>Disable frame discarting</display>
      <param value="1">Not working</param>
      <param value="0">Discarting</param>
    </item>
   </items>
  </module>
 </modules>
</device>
```

Listing 4.1: Addresses.xml fragment – ENUM format with parameters

The basic element of the configuration is an "item" element. One "item" is equal to one state or configuration entity. That means one register or even one bit according to the bitmask.

A "module" element represents one hardware module or just "item" elements whose location is the same. Generally, it is a list of "item" elements. The "module" element specifies a communication bus – currently supported are **CSBUS** and **MDIO**. All "item" elements specified in the "module" have to be located on the same bus.

Comstat(1) organizes the "item" elements into groups. Every "module" element with one "base address" creates one group. There is an attribute "header" of the "module" element. This attribute determines the text that is printed by *comstat(1)* as a header of the group.

Base address of the “module” is specified as an attribute of the “address” element. It is usually the address of the first “item” from the module. The “address” element can be used to clone the group of items. For example we have the COMBO card with two network interfaces. Every network interface has its own Statistical module in the firmware. These Statistical modules have the same structure, however they are located on different base addresses. In the configuration file, there is the structure of the Statistical module defined only once. The module for the second network interface is added by inserting new “address” element.

Every “item” element is supposed to have an offset address. It is the text child element of the “offset” element. The address of the hardware register consists of the base address and the offset address.

For items that consist of a subset of register bits, there is a “bitmask” element. It allows user to specify which bits of a register value are valid for the item. If “bitmask” is set to zero, all bits are valid.

The “item” element has the following possible child elements:

- **access** — The list of possible **access** values:

  RO (Read Only) — generally the state item,

  RW (Read–Write) — configuration item,

  DN (Do Nothing) — specifies items that are not read from hardware,

  CNT (Count) — compute the value from equation, see Section 4.3.4.

- **bitmask** — Bitmask is ANDed to the value that is read from register, the result is shifted right according to the bitmask, so that the first ’1’ bit of the bitmask is on the LSB. If the bitmask is equal to zero, all bits of value from register are valid for the item and there is no shifting[14].

- **display** — The text printed by *comstat(1)* as a label for the item.

- **name** — The name of the item used for addressing; *comstat(1)* works with the list of items according to item’s name.

- **format** — The value of **format** determines how the value of the item is printed by *comstat(1).* The list of possible **format** values:

  HEX — hexadecimal output,

  DEC — decimal output,

  BIN — binary output,

[14]Example: Let bitmask = 0x0C; value read from register = 0xE5: After AND operation, the value is 0x04. The result value after shifting is 0x01.

ENUM — the special type of enumeration, text equivalents of values is taken from "param" attributes,

HIST — histogram,

SWITCH — the output is "ON" for a non-zero value, otherwise "OFF",

INDICATE — the output is "UP" for a non-zero value, otherwise "DOWN",

DELAY — the output value is formatted as a time in seconds and nanoseconds,

HEADER — the output is text without any value, it is used as a title for the list of items.

VSWITCH, VINDICATE — SWITCH and INDICATE have their inverted form VSWITCH and VINDICATE. The output text of VSWITCH is "OFF" for a non-zero value (similarly VINDICATE).

- **offset** — The offset part of the register address; the final address is equal to $ba + offset$, where "ba" is the module's base address.

- **param** — The meaning of the **param** differs according to the selected **access** value. When **access** is CNT, **param** value specifies the equation. Otherwise, it contains the text equivalent of numeric value. For replacing numeric values by text equivalents from **param** elements, it is necessary to select the ENUM format. The list of optional **param** attributes:

  value — it is the number that will be replaced with given text,

  showtext — The value specifies whether to print text equivalent (if it is set to '1'),

  uimg — it is URL of image[15] that is inserted into the web page of comet.cgi (see Section 4.3.5).

- **repeat** — This attributes is used mostly for histograms – it specifies the amount of bins of the histogram.

- **size** — The Value of **size** element specifies the length of the item value. The list of possible **size** values:

  W (Word) — value is stored in one register (32 b value for CSBUS),

  D (Double word) — value is stored in two following registers (used for 64 b counters on CSBUS),

[15]Example: uimg="/images/capable.gif"

S (Swapped double word) — the same as D, but the order of registers is reversed (higher bits are in the register with lower address).

It is clear from Listing 4.1 that the “item” elements are the child of the “items” element. The “module” element was already described. It is wrapped by the element “modules”. The root element of the whole configuration tree is called “device”. The “device” element represents the COMET device. It is allowed to appear only once in the XML configuration document.

The “device” element currently contains only one element – “modules”, however the idea was to create some identification of the XML configuration file in the “device” subtree. This feature was not implemented yet, because it is not necessary. The “device” element must have exactly one child element “modules”.

Items from the configuration file are addressed in *comstat(1)* by their name. In order to address items correctly, it is recommended to use name of item that matches the regular expression (4.1). The item name should not begin with a digit and should contain only ASCII letters, digits and characters ’-’ and ’_’.

$$\texttt{\^{}[-\_a-Z][-\_a-Z0-9]*\$} \tag{4.1}$$

The complete example of the `addresses.xml` file is available on the enclosed CD. The path to the file is:

```
/comet/sw/comstat/addresses.xml
```

4.3.3 Process cycle

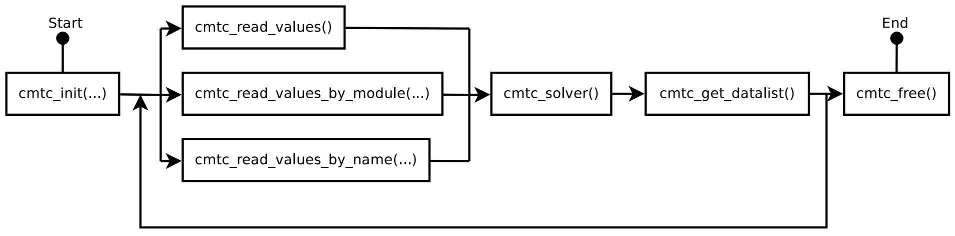


Figure 4.6: Comstat process cycle

The process cycle will be described with the help of Figure 4.6. This figure shows the usage of the *libcomet* module in *comstat(1)*. The initialization is done by *cmtc_init()* that requires one parameter – the path to the configuration file. The *cmtc_init()* function generates an inner representation of the item list. According to the item list, the library reads the values from hardware and creates a list of values (value list).

There are three types of item filter for reading. It is possible to read values of all “items” by *cmtc_read_values()*. The remaining two choices select the “items”, which are read. The names of the functions have the same prefix and the end of the name specifies a used filter. To read values of the “items” from one “module”, the name of the function ends with *_by_module*. The third possibility reads values of items with the name matching the regular expression. The function name has the suffix *_by_name*.

After reading values, it is useful to execute the COMET solver by the *cmtc_solver()* function. COMET solver goes through the value list and looks for the items with the CNT **access** (that means they were not solved yet). When an item is solved (computed), the COMET solver changes the **access** value to DN. Until the list of items is completely solved, the library computes CNT items one by one. The computation is mentioned in Section 4.3.4.

The resulting value list can be retrieved from *libcomet* by calling the *cmtc_get_datalist()* function

```
cmtc_value_list_t *         cmtc_get_datalist ();
```

that returns a one way linked list of `cmtc_value_t` structures. The declaration of `cmtc_value_t` is in Listing 4.2.

```
typedef struct cmtc_value {
        char *name;
        int error;
        uint64_t value;
        char *name_display;
        cmtc_param_t *param;
        cmtc_reg_access_t access;
        cmtc_val_format_t format;
} cmtc_value_t;
```

Listing 4.2: The structure cmtc_value_t

The most of entries of the *cmtc_value_t* structure are copied from the internal item list. On the other hand, value list does not contain information about physical addressing (base address, offset, etc.). Comparing to the item list, the value list contains the information about an error or success of the computation.

Comstat(1) can be used to print the list of sections. *Libcomet* contains the function *cmtc_get_namelist()*. This function returns the linked list of module names from the item list. The list element contains only two entries – *name* and *next*. The list must be freed manually.

The allocated memory of the value list is supposed to be manually freed with the library function *cmtc_free_datalist()*. When the work with the *libcomet* library module is already finished, there is the function that frees the inner item list – *cmtc_free()*.

4.3.4 Computation in comstat

When the **access** value of an item is equal to CNT, value is not read from hardware. It is computed for the item according to an equation given in the **param** element.

Comstat(1) expects the equation in a postfix format. One CNT item is shown in Listing 4.3. The value of the item from the listing is the average size of packets. It is computed according to Equation (4.2).

$$AVERAGE\_SIZE = \frac{ST\_SUM\_SIZE}{ST\_PKT\_REC + ST\_PKT\_CRC} \tag{4.2}$$

```
<item>
  <format>DEC</format>
  <access>CNT</access>
  <size>D</size>
  <bitmask>0x0</bitmask>
  <offset>0000</offset>
  <name>avg-pkt-size</name>
  <display>Average packet size</display>
  <param>ST_SUM_SIZE,ST_PKT_REC,ST_PKT_CRC,+,/</param>
</item>
```

Listing 4.3: Addresses.xml fragment – CNT access with equation in parameter

The equation consists of comma (',') separated names of items, user numeric constants and operators. The currently supported operators are '+', '-', '*', '/', '<' and '>'. The last two operators ('<' and '>') are the logical bitwise shift to left and right.

The COMET solver is a stack automaton. The maximal size of the stack is defined by the **SOLVER_STACK_SIZE** macro. This value is set to 5 by default. The stack size can be changed by reconfiguring and recompiling of the *libcomet* library. The configuration parameter is called:

```
--with-solver-stack=NUMBER
```

Listing 4.4 shows the way how to change the stack size.

```
make clean
./configure --with-solver-stack=10
make
```

Listing 4.4: Change of the COMET solver stack size

4.3.5 Compilation Output – COMSTAT and COMET.CGI

This section is about the compilation of *comstat(1)* and *comet.cgi* and has no connection with the *libcomet* library. The compilation of source codes in the `/comet/sw/comstat/` directory on enclosed CD is done by *make(1)* according to Makefile. This is the same behaviour as of the other COMET software utilities. However, the Makefile in `comstat` directory contains rules to generate two binaries – *comstat(1)* and *comet.cgi*.

The first output of compilation is *comstat(1)*. It is a console application for usage from the command line. *Comstat(1)* takes the input from given arguments. The list of parameters can be found in the manual page of *comstat(1)*.

The second output is *comet.cgi* that is a CGI application, The *comet.cgi* application creates the graphical user interface of COMET. It is described in Section 4.4.

The both binaries are generated from the same source codes. The main source code file (`comstat.c`) contains two branches – one for *comstat(1)* and one for *comet.cgi*. The compilation is controlled by internal precompiler macros: **HTMLOUTPUT** and **CGISCRIPT**. These two macros are passed to compiler in order to determine which branch to use. if the **HTMLOUTPUT** macro is defined, the output of binary is in HTML formal.

For generating HTML output, I have created my own *html* module. The *html* module consists of `html.c` and `html.h`. Even though *html* was not released as a library, it can be easily reused. All functions from *html* module have "html_" prefix. The *html* module creates an internal tree that is rendered.

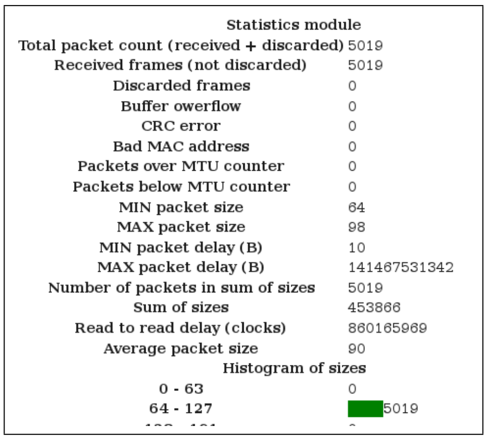
Despite the output format, *comet.cgi* differs from *comstat(1)* in the way of parameter passing. Defining **CGISCRIPT** macro, the CGI mode is enabled. The CGI application was developed and tested with the Apache 2 web server. This server stores the HTTP GET parameters into the **QUERY_STRING** environment variable. It was used to take parameters in *comet.cgi*.

4.4 User interface

COMET can be used from console via the *pcap2sze(1)*, *pcapedit(1)* and *comstat(1)* utilities. In addition, it has a simple web user interface for the *comstat* module. The Web pages are dynamically generated by *comet.cgi*. *Comet.cgi* is executed by the web server to generate a response to an HTTP request of a client.

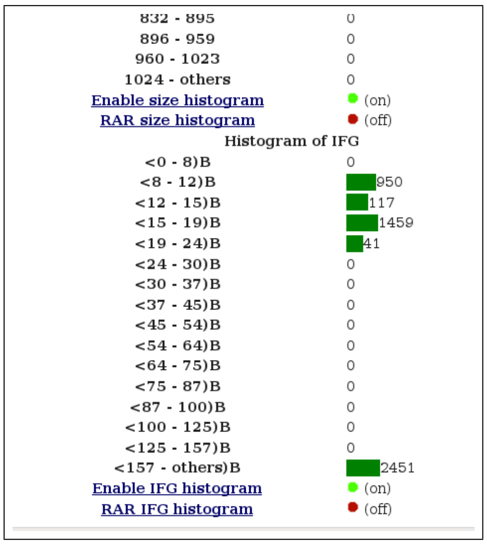
Figures 4.74.7 shows cropped screenshots of the *comet.cgi* HTML pages. The screenshots were captured on the Liberouter testing server during the test of firmware.

Figure 4.74.8(a) contains the data from Statistical module. It shows the basic statistical information about the testing traffic that was received. There is one computed item – "Average packet size".



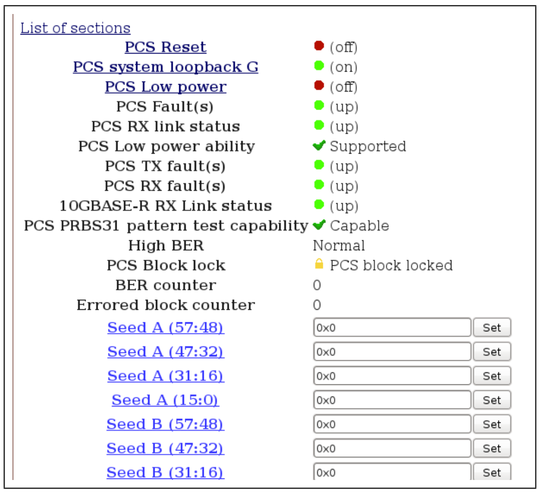
Statistics module	
Total packet count (received + discarded)	5019
Received frames (not discarded)	5019
Discarded frames	0
Buffer owerflow	0
CRC error	0
Bad MAC address	0
Packets over MTU counter	0
Packets below MTU counter	0
MIN packet size	64
MAX packet size	98
MIN packet delay (B)	10
MAX packet delay (B)	141467531342
Number of packets in sum of sizes	5019
Sum of sizes	453866
Read to read delay (clocks)	860165969
Average packet size	90
Histogram of sizes	
0 - 63	0
64 - 127	🟩5019

(a) Statistical module



832 - 895	0
896 - 959	0
960 - 1023	0
1024 - others	0
Enable size histogram	● (on)
RAR size histogram	● (off)
Histogram of IFG	
<0 - 8)B	0
<8 - 12)B	🟩950
<12 - 15)B	🟩117
<15 - 19)B	🟩1459
<19 - 24)B	🟩41
<24 - 30)B	0
<30 - 37)B	0
<37 - 45)B	0
<45 - 54)B	0
<54 - 64)B	0
<64 - 75)B	0
<75 - 87)B	0
<87 - 100)B	0
<100 - 125)B	0
<125 - 157)B	0
<157 - others)B	🟩2451
Enable IFG histogram	● (on)
RAR IFG histogram	● (off)

(b) IFG histogram



List of sections

PCS Reset	● (off)
PCS system loopback G	● (on)
PCS Low power	● (off)
PCS Fault(s)	● (up)
PCS RX link status	● (up)
PCS Low power ability	✓ Supported
PCS TX fault(s)	● (up)
PCS RX fault(s)	● (up)
10GBASE-R RX Link status	● (up)
PCS PRBS31 pattern test capability	✓ Capable
High BER	Normal
PCS Block lock	PCS block locked
BER counter	0
Errored block counter	0
Seed A (57:48)	0x0 Set
Seed A (47:32)	0x0 Set
Seed A (31:16)	0x0 Set
Seed A (15:0)	0x0 Set
Seed B (57:48)	0x0 Set
Seed B (47:32)	0x0 Set
Seed B (31:16)	0x0 Set

(c) PCS

Figure 4.7: Fragments of COMET screenshots

The sample of histogram is shown in Figure 4.74.8(b). We can see amounts of Ethernet frames that were received. The frames are divided into bins of histogram according to the inter-frame gap. The histogram values can be reset by the RAR IFG histogram (RAR is the abbreviation for Reset after read).

The third figure with screenshot (Figure 4.74.8(c)) shows the items of PCS. This screenshot demonstrates the possibilities of item formats. In PCS group of items there is a lot of configurable items. *Comet.cgi* allows user to fill in the new value and change the value of the item in hardware. The

HTML form with an input text accepts a number in decimal or hexadecimal. Clicking on the hypertext link on the label causes the inversion of the current value. In addition, this screenshot shows the result of usage of the ENUM **format**. I specified the text and icon for some items such as PCS Block lock and PRBS31 test capability. We can find the examples of SWITCH and INDICATE values of **format** – PCS reset and PCS Fault(s).

When the HTML form is submitted *comet.cgi* generates automatic redirection[16]. It is done by sending special HTTP header from server. The header is called "Location". The redirection is handled by the *html* module before rendering an HTML page. The inversion of value by clicking on the item link causes the same reaction occurs.

In order to determine the redirect URL, an environment variable is used. It is the **HTTP_REFERER** variable that contains the URL of previous location. This variable is set by web server.

4.5 NETCONF plug-in

COMET supports the NETCONF protocol by implementing a plug-in for Netopeer configuration system. The Netopeer plug-in is a dynamically loadable file that implements given interface from *dev_modules_interface.h*. This header file contains an API for *netopeer-server*. The source codes of NETCONF plug-in can be found in the */comet/sw/netcomet* directory on enclosed CD. The main file is *netopeer-cfgcomet.c*.

The Netopeer system expects a plug-in to have three functions implemented. The first two of these functions are for initialization and finalization (*init_plugin()* and *close_plugin()*). The third function is the most important. Its name is *execute_operation()* and it is internally called at every request the server accepts. There is a chain of comparisons in this function to determine the NETCONF request operation. The basic operations are defined in [17, RFC6241]. The device model can define its own operations as new RPC methods. I have added **send-traffic** and **reset-counters** into the COMET configuration model. The COMET configuration model is placed into Appendix D.

When it is known, what operation is being executed, the specific "execute" function is internally called. This mechanism creates the structure of the plug-in source codes. The structure of the *netopeer-cfgcomet.c* file is quite the same as in Netopeer example plugin. The COMET plug-in, as Netopeer plug-in does, uses the *libnetconf* library. *Libnetconf* implements the functionality of the NETCONF protocol. The manipulation with *datastores* is also performed in the library.

[16] HTML form is sent by an HTTP POST method; without the redirection, browser would resent the value during every refresh of the page; in addition, the redirection is the valid reaction on the POST method.

NETCONF RFC defines three *datastores* – *running*, *startup* and *candidate*. The datastore is a conceptual place to store and access information. *Datastores* have the following meaning:

running — a configuration datastore holding the complete configuration currently active on the device,

startup — a configuration datastore holding the configuration loaded by the device when it boots,

candidate — a configuration datastore that can be manipulated without impacting the device's current configuration and that can be committed to the running configuration datastore.

Netopeer repository is an implementation of *datastore*. At the start-up phase, the COMET plug-in checks the existence of the valid *startup* datastore. If it does not exist, plug-in creates the new instance of *startup* datastore and fills it with initial values. The *startup* datastore is then copied to the running repository. The NETCONF method **<get-config>** returns the content of the *running* datastore if no else is given as the source repository. That means the *libnetconf* function for obtaining datastore content was used as the Netopeer plug-in example does.

The different situation is in the **<get>** method handler. This method should return the configuration including state information. The state information has to be read from hardware, in order to be up-to-date. The implementation of the COMET plug-in works with generated templates which are filled with the latest values.

There are two templates for COMET. I created the *resptempl.py* plug-in for *pyang* system to generate these templates. This *pyang* plug-in is in the `netcomet` directory on enclosed CD. The *resptempl.py* works as another *pyang* plug-in *tree.py*. *Tree.py* was distributed with pyang. It iterates over the configuration model tree and prints the elements as an ASCII tree. The output of *resptemp.py* is an XML template for configuration or state information model tree. I created another python script *resptempl2c.py* that processes the generated XML templates:

```
comet_xml_state.xml
comet_xml_config.xml
```

The output of *resptempl2c.py* is the file with a definition of constant string in the C language (`const char *`). For COMET, there are two files with definition:

comet_xml_config.c with configure information template,
comet_xml_state.c with state information template.

When we have templates generated from the model, they can be used in the COMET NETCONF plug-in. The templates are compiled with the rest of source codes. The variables from these files are used to load an XML tree with template. The *comet_xml_config.c* is used only during the initialization, when no *startup* datastore exists. The *comet_xml_state.c* template is refilled during every **<get>** request. The filling of the template is based on the XML tree traversing. It is a recursive function, which is looking for leaves of the tree. When function gets to the leaf, it tries to get value with corresponding name from *libcomet*. If the value is found, it is appended as a text child of the leaf.

The *pcap2sze(1)* functionality is also supported over the NETCONF protocol. In the plug-in, there is a definition of the default path, where PCAP files should be located. It is defined by **PCAP_FILEDIR** macro and it can be changed using *configure* script:

```
make clean
./configure --with-pcap-dir=/path/to/directory
make
```

The COMET model defines the XML structure of the "sender" configuration. It contains a list of PCAP files. Every PCAP file has a number that is an index of file. One index is "active". It was chosen, that plug-in searches for files matching the regular expression 4.3.

```
#define PCAP_REGEXP_STR "^[-_ a-Z0-9]*\\.pcap$"        (4.3)
```

When creating *startup* datastore, the subtree of the "sender" element from the template is completely deleted. It is more efficient and easier to create new subtree then appending data to existing one on the right places. The names of PCAP files are filled in relative path. It is expected to add **PCAP_FILEDIR** in time of need – when the file should be opened.

Every time any NETCONF request modifies the *running* configuration, plug-in updates the values in hardware. For this task, the function *update_hw* was written. This function gets the current running configuration and writes everything to hardware. It is done by iterating over the configuration tree like during template filling.

If **send-traffic** request occurs, the plug-in takes one argument from the request message. The argument specifies what mode of transmission was chosen. The COMET plug-in supports these modes according to the model:

full-speed takes the user configured **copies** and sends the traffic with the interval set to zero

user-speed sends traffic at the user configured speed

interval-based takes the user configured **interval** and the **first timestamp**

Chapter 5

Testing and Verification

All software tools were checked with the Valgrind [7] system. The homepage of the Valgrind project gives this definition:

> Valgrind is an instrumentation framework for building dynamic analysis tools. There are Valgrind tools that can automatically detect many memory management and threading bugs, and profile your programs in detail. You can also use Valgrind to build new tools.

```
==6451==
==6451== HEAP SUMMARY:
==6451==     in use at exit: 0 bytes in 0 blocks
==6451==   total heap usage: 12,787 allocs, 12,787 frees,
688,492 bytes allocated
==6451==
==6451== All heap blocks were freed -- no leaks are possible
==6451==
==6451== ERROR SUMMARY: 0 errors from 0 contexts
(suppressed: 6 from 6)
--6451--
--6451-- used_suppression:      6 dl-hack3-cond-1
==6451==
==6451== ERROR SUMMARY: 0 errors from 0 contexts
(suppressed: 6 from 6)
```

Figure 5.1: The output of Valgrind – result of the test of *comstat(1)*

All errors and memory leaks discovered by Valgrind were fixed during the development. The results of tests of the written code were without errors and memory leaks. The example of Valgrind output is in Figure 5.1. Valgrind was executed by verbose parameter (`-v`) and with the full check for memory leaks (`--leak-check=full`). The *comstat(1)* application was executed to print all items from statistical module (`comstat -s 3`) in Figure 5.1.

5.1 Static Analysis of Code

In addition to dynamic analysis that was done by Valgrind, the COMET source codes were checked by static code analysis method. The static code analysis is performed without executing program. The software that was used for static code analysis is *Stanse* [28].

Stanse was used to discover bugs in Linux Kernel. It can run multiple checkers to find errors such as memory allocation errors (null pointers, memory leaks, dangling pointers). *Stanse* can find errors that can be described by state automaton.

During the analysis, I discovered “NULL pointer dereference” problem in the source code of *netcomet*. The problem was caused by missing condition after memory allocation. The problem was successfully fixed.

I wrote a BASH script for testing all parts of COMET software. The content of the script is shown in Listing 5.1.

```
stansedir=stanse-1.1.2/
mkfls="packet_gen/Makefile packet_gen/sw/libcomet/↩
    Makefile packet_gen/sw/netcomet/Makefile"

for mkfl in $mkfls; do
        curdir=$PWD
        echo $mkfl
        cd $(dirname $mkfl)
        make clean
        cd $curdir
        cd $stansedir
        java -jar stanse.jar  -c AutomatonChecker:data↩
            /checkers/AutomatonChecker/memory.xml --↩
            makefile "$curdir/$mkfl"
        cd $curdir
done
```

Listing 5.1: BASH script for static code analysis (Stanse)

5.2 Sender and Receiver

The *pcap2sze(1)* and *comstat(1)/comet.cgi* were deployed on the Liberouter development server. They were used for testing of COMET firmware during the development.

In the combination with the Ethernet tester Spirent, there were three testing scenarios. The testing of the sender and receiver firmware part was done in cooperation with Bc. Pavel Benáček, the author of the COMET firmware [8].

The aim of the tests of COMET software was to verify that the presented values are equal to real values. For the receiver part, the incoming traffic is analysed by firmware. However, we checked the values that were read from firmware by *comstat(1)*.

On the other direction, we tested the *pcap2sze(1)* program. It was used to send testing data of various (known) sizes. We used two different targets of transmission. On the receiver side we had the COMET device or the Spirent Ethernet tester (AX/4000). During the test, we compared the amount of frames that were sent and that were received.

The test of histogram values used *comstat(1)* and a BASH script that can read the histogram values. The script was written by the author of COMET firmware and it uses the Liberouter command line utilities[17]. Values read by *comstat(1)* were compared with the output of this script. All values were equal therefore the algorithm of histogram reading was well implemented in *comstat(1)*.

5.2.1 COMET Sender – COMET Receiver

This type of test was run locally on one development server. The server was loaded with COMET firmware and COMET software tools were installed. This test was based on enabled hardware loopback in the phyter (PCS system loopback G). The COMET module Blackhole was also enabled – the module for discarding incoming packets[18].

The transmission was done by *pcap2sze(1)*. Information about received traffic was read by *comstat(1)*. We checked the amount of the incoming data – whether the received and sent packets amount are equal. It is obvious that the COMET firmware and software did not lose or drop any packet.

In addition, we checked whether the data of the traffic were corrupted. There is the Liberouter version of *libpcap*, so that we were able to execute *tcpdump(8)* on the COMBO interface and save the traffic. Then, it was possible to compare incoming and outgoing data. *Pcapedit(1)* was used to compare PCAP file content without timestamps and PCAP headers. All data were received correctly and no packet was corrupted.

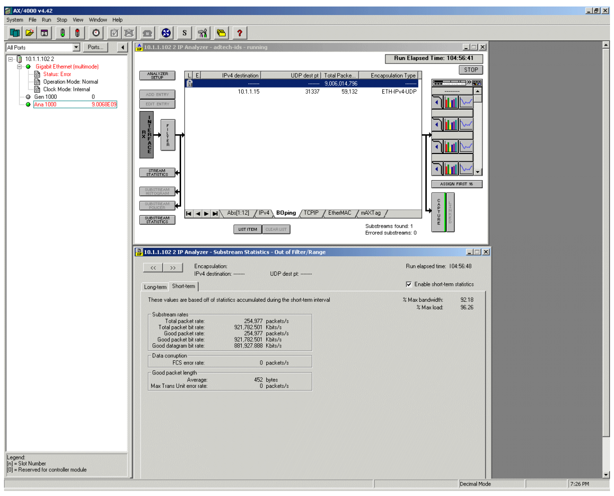
[17]The script is not efficient enough because it executes *csbus(1)* Liberouter utility for every histogram bin. That means there is an initialization of the communication interface for every histogram bin. It is recommended to use *comstat(1)* instead. (The execution time of the script and *comstat(1)* were measured by *time(1)*. The results were 0m1.488s for the script and 0m0.073s for *comstat(1)*.)

[18]If the packets were not discarded, they would stay in the incoming queue. Blackhole frees the queue but has no influence on statistics.

5.2.2 COMET Sender – Spirent Receiver

This test was similar to 5.2.1. The difference was on the side of the receiver. The COMET tester was connected to the Spirent Ethernet tester. The loopback from the test in Section 5.2.1 was disabled.

The amount of incoming packets and the packet sizes were compared. All incoming packets were received and counted. The average packet size was computed correctly.



10.1.1.102 2
Gigabit Ethernet (multimode)
Status: Error
Operation Mode: Normal
Clock Mode: Internal
Gen 1000 0
Ana 1000 9.0068E09

Legend:
[n] = Slot Number
[0] = Reserved for controller module

10.1.1.102 2 IP Analyzer - adtech-ids - running

Run Elapsed Time: 104:56:41

L	E	IPv4 destination	UDP dest pt	Total Packe...	Encapsulation Type
		--------	--------	9,006,014,796	--------
		10.1.1.15	31337	59,132	ETH-IPv4-UDP

Substreams found: 1
Errored substreams: 0

10.1.1.102 2 IP Analyzer - Substream Statistics - Out of Filter/Range

Encapsulation:
IPv4 destination: -------- UDP dest pt: --------
Run elapsed time: 104:56:48
☑ Enable short-term statistics

These values are based off of statistics accumulated during the short-term interval

% Max bandwidth: 92.18
% Max load: 96.26

Substream rates

Total packet rate:	254,977	packets/s
Total packet bit rate:	921,782.501	Kbits/s
Good packet rate:	254,977	packets/s
Good packet bit rate:	921,782.501	Kbits/s
Good datagram bit rate:	881,927.888	Kbits/s

Data corruption

FCS error rate:	0	packets/s

Good packet length

Average:	452	bytes
Max Trans Unit error rate:	0	packets/s

Figure 5.2: Spirent AX/4000 control software – screenshot for illustration

5.2.3 COMET Receiver – Spirent Sender

The Spirent tester can generate network traffic with various characteristics. We used this tester to check the behaviour of COMET, when the traffic contains a lot of small packets or the large packets.

We run short tests and long tests. The short test was based on fixed amount of packets – 100000. The sizes of packets were: 64, 65, 127, 128, 512, 513, 768, 1023, 1024, 1280 and 1518 bytes. The long test was based on sending traffic at full speed for an hour.

Packets of all sizes, that were sent, were received correctly. The distribution of packet sizes that was seen in histogram corresponded to the distribution that was set in the Spirent tester.

5.2.4 Test of Packet Delay Histogram

This section is focused on one of the tests of the Packet delay histogram unit. During the tests, the system loopback was enabled.

I used the *pcap2sze(1)* utility to send the traffic with given intervals in relative mode. That means the first packet was sent immediately. Every packet was repeated so that the transmission took two minutes. Table 5.1 shows the amount of packets that were sent. The first column contains the delay that was set. The second and the third column contain the amount of packets.

Table 5.1: Amounts of sent and received packets depending on the interval. The transmission was done by *pcap2sze(1)*. Data for this table were taken from [8].

Interval	**Sent packets**	**Received packets**
2.5 ms	48010	48010
400 μs	300010	300010
80 μs	1500010	1500010
10 μs	12000010	12000010
2 μs	60000010	60000010
Total amount	73848050	73848050

Traffic was sent by these commands:

```
./pcap2sze -f ~/test.pcap  -v -t0s2500000 -i 0 -r4800 -F1
./pcap2sze -f ~/test.pcap  -v -t0s400000 -i 0 -r30000 -F1
./pcap2sze -f ~/test.pcap  -v -t0s80000 -i 0 -r150000 -F1
./pcap2sze -f ~/test.pcap  -v -t0s10000 -i 0 -r1200000 -F1
./pcap2sze -f ~/test.pcap  -v -t0s2000 -i 0 -r6000000 -F1
```

The *test.pcap* file contains ten frames with average size of approximately 100 B. The result of the test is shown in Figure 5.3. In the relative mode, the delay of packets is measured between ends of the two following packets. That is why the figure contains some frames in the bin with range 0–1μs. The description can be found in more detail in [8]. No packets were lost, the delays were correct.

5.3 NETCONF Plug-in

In order to test the Netopeer plug-in, there is an application called *plugin-tester*. The application expects the name of one or more plug-ins and input XML files as parameters. The input XML files contain NETCONF requests.

The *plugin-tester* was used for testing COMET plug-in. It was more comfortable than repeatedly connecting to Netopeer server via *netopeer-client*. *Netopeer-client* connects to server using SSH protocol. *Plugin-tester* is much faster because it does not need to establish an SSH connection.

With *plugin-tester*, the expected response can be compared with the output of plug-in. I have prepared a set of testing requests. The files with requests are placed on enclosed CD. These are meanings of sample requests:

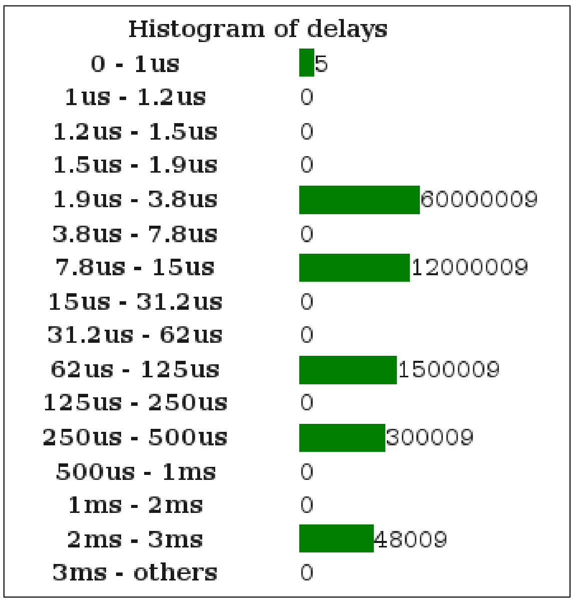


Figure 5.3: The result of the test of Packet delay histogram

1. **test.xml** — the basic test of Netopeer,
2. **get.xml** — <get> request; returns merged state and configuration information,
3. **get-config.xml** — <get-config> returns configuration information,
4. **get-config-filter.xml** — test of <get-config> with filter set on *sender*,
5. **edit-config.xml** — change of some values,
6. **edit-config-replace.xml** — the same behaviour as edit-config.xml,
7. **setup-sender.xml** — sets the configuration of sender,
8. **send.xml** — RPC method that begins a transmission according to configuration,
9. **reset.xml** — RPC method, that resets counters.

Functionality of the plug-in was verified using these sample requests. I have created a simple BASH script for testing automatically. The script reads the files with the name ending by *tst*. These files contain the list of the sample requests that are used for the test. The script executes *plugin-tester* with the parameters and the output of the program is saved into the *out* directory.

I have checked the output according to the configuration data model. When the response of the plug-in was correct, I moved the output file into the *exp* directory. When the test is run again, the output is compared with expected output. Therefore, the tests can be easily repeated after changes of the plug-in source codes and the functionality can be checked.

Conclusion

My task was to design and implement a control software for the new Ethernet tester called COMET. This system is based on COMBO card "Programmable hardware". This text concluded the process of development and described the inner organization of the resulting software utilities *pcap2sze(1)*, *comstat(1)* and *pcapedit(1)*.

During the work on this project, I have created the *libcomet* library that contains the most of the functionality of the software tools. This approach allowed me to study the process of shared libraries development.

When the *libcomet* library was implemented, I have studied a creation of RPM packages. The results of my work on the *libcomet* library are the *libcomet-0.9.0-1.x86_64.rpm* package and a development version of the package.

In order to describe the location of registers, I have designed the configuration file format based on the XML technology.

I have dealt with the configuration and management protocols, especially about the NETCONF protocol. I understood the reason of creation of the NETCONF protocol the way how this protocol works. I have studied the YANG modeling language and then I was able to modify the COMET configuration data model.

In collaboration with the author of firmware, we were able to create functional Ethernet tester. I have integrated the new system into the Netopeer configuration system by the dynamically loadable file *netopeer-cfgcomet.so*.

According to test results, the COMET tester can be deployed and used at the speed of 10 Gbps. During this year it is planned to migrate the COMET system to 40/100 Gbps. The software part that I worked on, is ready for the migration – it will work without any changes.

Future Work on COMET

All tasks have been fulfilled and the COMET software is complete and working. However, there are some opportunities to improve the current implementation. They are planned as a possible continuation of this project.

At first there is a need to implement the support of I^2C. It will be done by extending the *addresses.xml* file format (that means a modification of configuration parser) and inserting some code into two functions of *libcomet* – *read_values()* and *write_config()*.

For a future work, it is planned to improve the web user interface. The current *comet.cgi* CGI application will not generate the target HTML code. Instead of HTML, the output will be in an XML or a JSON format. The presentation tier will be made by currently used technologies for the web pages development. The values will be updated on the web page using JavaScript – AJAX technology. This will lead to more flexible user interface and more possibilities to redesign the web pages.

The potential work is also on the *pcap2sze(1)* utility. This application expects the number of copies of one frame for using the full speed mode. Users usually need to generate network traffic for the specific time. It would be useful for users to specify how long should the transmission take and let the application to determine all the other parameters.

Bibliography

[1] An extensible YANG validator and converter in python. 2012. Available at WWW: `<http://code.google.com/p/pyang/>`

[2] Common Object Request Broker Architecture - Wikipedia, the free encyclopedia. 2012. Available at WWW: `<http://en.wikipedia.org/wiki/CORBA>`

[3] FlowMon - Comprehensive Solution for NetFlow Monitoring - INVEA-TECH. 2012. Available at WWW: `<http://www.invea-tech.com/products-and-services/flowmon>`

[4] Management information base - Wikipedia, the free encyclopedia. 2012. Available at WWW: `<http://en.wikipedia.org/wiki/Management_information_base>`

[5] Netconf Central. 2012. Available at WWW: `<http://netconfcentral.org>`

[6] Structure of Management Information - Wikipedia, the free encyclopedia. 2012. Available at WWW: `<http://en.wikipedia.org/wiki/Structure_of_Management_Information>`

[7] Valgrind Home. 2012. Available at WWW: `<http://valgrind.org/>`

[8] BENÁČEK, Pavel: *Ethernet tester for high-speed networks.* Master's thesis, Faculty of Information Technology, Czech Technical University, 2012, written in Czech.

[9] Bjorklund, M.: YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF). RFC 6020 (Proposed Standard), Oct. 2010. Available at WWW: `<http://www.ietf.org/rfc/rfc6020.txt>`

[10] Case, J.; Mundy, R.; Partain, D.; etc.: Introduction to Version 3 of the Internet-standard Network Management Framework. RFC 2570 (Informational), Apr. 1999, obsoleted by RFC 3410. Available at WWW: `<http://www.ietf.org/rfc/rfc2570.txt>`

[11] CESNET, z.s.p.o: CESNET, z.s.p.o. 1996–2012. Available at WWW: <http://www.cesnet.cz>

[12] CESNET, z.s.p.o: COMBO6 | www.liberouter.org. 2002-2009. Available at WWW: <http://www.liberouter.org/card_combolxt.php>

[13] CESNET, z.s.p.o: NetCOPE | www.liberouter.org. 2002-2009. Available at WWW: <http://www.liberouter.org/netcope/>

[14] CESNET, z.s.p.o: Programmable hardware | www.liberouter.org. 2002-2009. Available at WWW: <http://www.liberouter.org>

[15] Dimitri van Heesch: Doxygen. 2012. Available at WWW: <http://www.stack.nl/~dimitri/doxygen/>

[16] Enns, R.: NETCONF Configuration Protocol. RFC 4741 (Proposed Standard), Dec. 2006, obsoleted by RFC 6241. Available at WWW: <http://www.ietf.org/rfc/rfc4741.txt>

[17] Enns, R.; Bjorklund, M.; Schoenwaelder, J.; etc.: Network Configuration Protocol (NETCONF). RFC 6241 (Proposed Standard), June 2011. Available at WWW: <http://www.ietf.org/rfc/rfc6241.txt>

[18] Free Software Foundation, Inc: Autoconf - GNU Project - Free Software Foundation (FSF). Available at WWW: <http://www.gnu.org/software/autoconf/>

[19] Free Software Foundation, Inc: Automake - GNU Project - Free Software Foundation (FSF). Available at WWW: <http://www.gnu.org/software/automake/>

[20] Free Software Foundation, Inc: Make - GNU Project - Free Software Foundation (FSF). Available at WWW: <http://www.gnu.org/software/make/>

[21] HEDSTROM, Brian; Akshay Watwe; Siddharth Sakthidharan: *Protocol Efficiencies of NETCONF versus SNMP for Configuration Management Functions*. Master's thesis, University of Colorado, 2011.

[22] KREJČÍ, R.: Konfigurace síťových zařízení protokolem NETCONF [online]. 2007. Available at WWW: <http://theses.cz/id/a7q8qk/>

[23] KREJČÍ, Radek: libnetconf - NETCONF library in C. 2012. Available at WWW: <http://code.google.com/p/libnetconf/>

[24] KREJČÍ, Radek: Remote configuration system using NETCONF protocol. 2012. Available at WWW: <http://code.google.com/p/netopeer/>

[25] LHOTKA, Ladislav: Configuring Network Devices with NETCONF and YANG. 2011. Available at WWW: <http://www.xmlprague.cz/2011/presentations/ladislav-lhotka-netconf-yang.pdf>

[26] MICHALÍK, M.: Testovanie paketového filtru NIFIC v prostredí reálnej siete Bakalářská práce, Masarykova univerzita, Fakulta informatiky. 2011. Available at WWW: <http://theses.cz/id/f203ib/>

[27] Schoenwaelder, J.: Overview of the 2002 IAB Network Management Workshop. RFC 3535 (Informational), May 2003. Available at WWW: <http://www.ietf.org/rfc/rfc3535.txt>

[28] SLABÝ, Jiří: Stanse - Static Analysis Framework for C code. 2009. Available at WWW: <http://stanse.fi.muni.cz/>

[29] Spirent Communications: Spirent - A leader in test, measurement and service assurance solutions. 2012. Available at WWW: <http://www.spirent.com/>

[30] The Apache Software Foundation: The Apache HTTP Server Project. 2012. Available at WWW: <http://httpd.apache.org/>

Appendix A

Acronyms

GUI Graphical user interface

CLI Command line interface

XML Extensible markup language

PCAP Packet Capture - file format for storage of network traffic

FPGA Field-programmable gate array

COMET COMbo Ethernet Tester

CORBA Common Object Request Broker

IDL Interface definition language

LSB Least significant bit

MSB Most significant bit

MIB Management Information Base

SMI Structure of Management Information

SFP Small Form-factor Pluggable transceiver

XFP 10 Gigabit Small Form-factor Pluggable transceiver

ISO/OSI International Organization for Standardization / Open Systems Interconnection

RPC Remote Procedure Call

Appendix B

Installation Manual

This installation manual was written and tested on the Scientific Linux distribution, used on Liberouter team's servers. The similar installation procedure could be done on other RPM based distributions such as Red Hat or Fedora with YUM package manager.

Before installation, the CESNET repository should be added into the YUM configuration. Create new file /etc/yum.repos.d/liberouter.repo

```
[Liberouter-stable]
name=Liberouter - Tools for Monitoring and ↩
    Configuration
baseurl=http://homeproj.cesnet.cz/rpm/liberouter/↩
    stable/$basearch
enabled=1
gpgcheck=1
gpgkey=http://homeproj.cesnet.cz/rpm/liberouter/RPM-↩
    GPG-KEY-liberouter


[Liberouter-devel]
name=Liberouter - Tools for Monitoring and ↩
    Configuration
baseurl=http://homeproj.cesnet.cz/rpm/liberouter/devel↩
    /$basearch
enabled=1
gpgcheck=1
gpgkey=http://homeproj.cesnet.cz/rpm/liberouter/RPM-↩
    GPG-KEY-liberouter
```

Listing B.1: Liberouter repository configuration

B.1 Dependencies

The COMET project depends on some Liberouter packages, *libxml2* and *libxml2-devel* packages. This is the list of Liberouter packages:

- libcommlbr
- libsze2
- libcombo
- libcombo-devel
- libsze2-devel
- libcommlbr-devel
- libpcap-devel

B.2 Libcomet

Installation from repository (for x86_64 architecture) is done by:

```
yum install libcomet
```

In case it is needed to compile *libcomet* from source codes, copy the files from the CD directory and type:

```
./configure
make
make install
```

For RPM package creation, run

```
make rpm
```

instead of “make install”.

B.3 COMET

After the successful installation of *libcomet* and dependencies, all tools from the COMET project can be compiled.

The project uses Autotools, so the compilation is done by executing in the root directory:

```
./configure
make
make install
```

B.4 COMET Start-up

After successful installation, COMET must be started. To start and initialize the COMET tester, use the *comet-boot.sh* script. *Comet-boot.sh* is located in */fw* directory on the CD.

B.5 Netcomet

Netcomet depends on the latest version of *libnetconf* and *libcomet*. It is recommended to install the compiled version of *libnetconf* from the Netopeer repository.

Netcomet (the COMET Netopeer plug-in) has a separate *configure* script. The compilation is done using *make(1)*.

When we have the plug-in compiled, the BASH script **install.sh** in the */comet/sw/netcomet/config* directory should prepare everything that is needed.

Appendix C

Netopeer Architecture Scheme

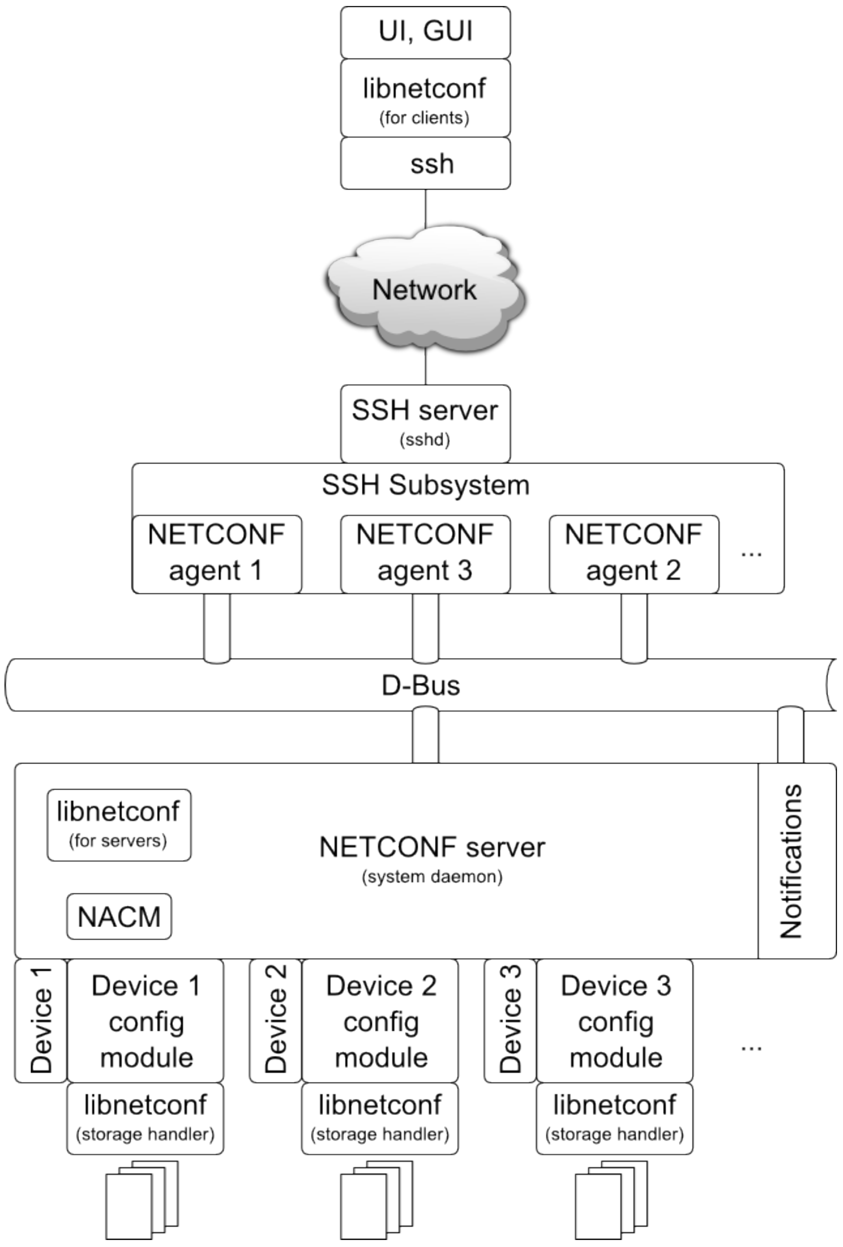


Figure C.1: The scheme of the Netopeer 2 project, published on the official project web sites [24].

Appendix D

COMET configuration model

```
module comet-tester {

  namespace "http://cesnet.cz/ns/yang/comet-tester";

  prefix "ct";

  import ietf-yang-types {
    prefix "yang";
  }

  import ietf-inet-types {
    prefix "inet";
  }

  /* Metadata */
  organization
    "CESNET, z. s. p. o.";
  contact
    "cejkat@cesnet.cz";
  description
    "This YANG module defines the data model for the ↩
       Combo Ethernet
     Tester (COMET).

     It is able to retrieve information from the COMET↩
         firmware,
     send stored testing network traffic and obtain
     statistical information about incoming traffic.
    ";

  revision 2012-05-04 {
    description
      "Master thesis revision.";
  }
  grouping enabled-switch {
    description
      "This grouping defines a switch which is used ↩
         for
       enabling/disabling individual COMET modules.";
    leaf enabled {
      type boolean;
      default "true";
    }
  }
  grouping histogram64 {
    description
      "This grouping defines reusable contents for ↩
         histograms with
       64-bit counters for each bin.

       Histograms are statistics, hence 'config false↩
          '.

       This grouping is supposed to be used in a ↩
          container
       representing the entire histogram. Such a ↩
          container node
       should provide, in its 'description' ↩
          substatement, parameters
       necessary for interpreting the histogram counts↩
          , i.e. at
       least minimum and maximum values and the total ↩
          number of bins.
      ";
    list bin {
      key "seq-number";
      config "false";
      max-elements "256";
      description
        "Each entry in this list gives the count of ↩
           items in a single
         histogram bin identified by its sequence ↩
           number.

          Maximum number of bins is 256.

          Sequence numbers of the entries must be ↩
              increasing. However,
          entries for one or more bins may be missing -↩
              count of that
          bin is then considered to be zero.
        ";
      leaf seq-number {
        type uint8;
      }
      leaf count {
        description
          "Number of values in the corresponding ↩
              histogram bin.";
        type uint64;
      }
    }
  }
  container comet-testers {
    list comet-tester {
      key "interface-id";
      leaf interface-id {
        type uint16 {
          range "0..max";
        }
      }
      container statistics {
        uses enabled-switch;
        container data {
          config "false";
          leaf pkt-sum {
            mandatory "true";
            must ". = ../pkt-rec + ../pkt-dis";
            description
              "Total packet count (received + ↩
                  discarded).";
            type yang:zero-based-counter64;
          }
          leaf pkt-rec {
            mandatory "true";
            description
              "Counter of received frames.";
            type yang:zero-based-counter64;
          }
          leaf pkt-dis {
            mandatory "true";
            description
              "Counter of discarded frames.";
            type yang:zero-based-counter64;
          }
          leaf pkt-bo {
            mandatory "true";
            description
              "Counter of packets causing buffer ↩
                 overflow.";
            type yang:zero-based-counter64;
          }
          leaf pkt-crc {
            mandatory "true";
            description
              "Counter of packets with CRC error.";
            type yang:zero-based-counter64;
          }
          leaf pkt-mac {
            mandatory "true";
            description
              "Counter of packets with bad MAC address↩
                 .";
            type yang:zero-based-counter64;
          }
          leaf pkt-bmtu {
            mandatory "true";
            description
              "Counter of packets whose size is less ↩
                 than MTU.";
            type yang:zero-based-counter64;
          }
          leaf pkt-omtu {
            mandatory "true";
            description
              "Counter of packets whose size is ↩
                 greater than MTU.";
            type yang:zero-based-counter64;
          }
          leaf min-size {
            mandatory "true";
            description
```

```
              "Minimum size.";
            type uint32;
          }
          leaf max-size {
            mandatory "true";
            description
              "Maximum size.";
            type uint32;
          }
          leaf min-delay {
            mandatory "true";
            description
              "XGMII minimum delay.";
            type uint64;
            units "bytes";
          }
          leaf max-delay {
            mandatory "true";
            description
              "XGMII maximum delay.";
            type uint64;
            units "bytes";
          }
          leaf avg-bitrate {
            mandatory "true";
            description
              "Current average bitrate.";
            type uint64;
            units "bps";
          }
          leaf avg-pkt-size {
            mandatory "true";
            description
              "Current average packet size.";
            type uint64;
            units "bytes";
          }
          container packet-sizes {
            description
              "Histogram of observed packet sizes.";
            uses histogram64;
          }
          container packet-gaps {
            description
```

```
              "Histogram of observed interpacket gaps.↩
                 ";
            uses histogram64;
          }
        }
      }
      container sender {
        uses enabled-switch;
        leaf interface {
          description
            "Output interface.";
          type uint16;
        }
        leaf timestamp {
          description
            "Timestamp of the first packet.";
          type uint64;
          units "nanoseconds";
        }
        leaf start-delay {
          description
            "Delay of the first packet.";
          type uint64;
          units "nanoseconds";
        }
        leaf interval {
          description
            "Time interval between packets.";
          type uint64;
          units "nanoseconds";
        }
        leaf copies {
          description
            "Number of copies of each packet.";
          type uint32;
        }
        leaf speed {
          description
            "Transmission speed.";
          type uint64;
          units "bits/s";
        }
        leaf hw-timestamp {
          description
```

```
            "If true, get the first timestamp from ↩
               hardware.";
          type boolean;
        }
        container pcap-files {
          description
            "List of uploaded pcap files. At any ↩
               moment, only one of
            them is active.";
          leaf active {
            description
              "Reference to the number of the active ↩
                 pcap file.

              No file is active if this leaf is ↩
                 missing.
              ";
            type leafref {
              path "../pcap-file/number";
            }
          }
          list pcap-file {
            description
              "URL of the PCAP file with a sequence ↩
                 number.

              Numbers must always form an increasing ↩
                 sequence.
              ";
            key "number";
            leaf number {
              type uint8 {
                range "1..max";
              }
            }
            leaf url {
              type inet:uri;
            }
          }
        }
        container data {
          config "false";
          leaf status {
            mandatory "true";
            description
              "Operational status of the sender module↩
                 .";
            type enumeration {
              enum sending;
              enum waiting;
            }
          }
        }
      }
      container output-packet-checker {
        uses enabled-switch;
      }
      container input-packet-checker {
        uses enabled-switch;
        container data {
          config "false";
          leaf inord {
            mandatory "true";
            description
              "Counter of packets arriving in order.";
            type yang:zero-based-counter64;
          }
          leaf outord {
            mandatory "true";
            description
              "Counter of packets arriving out of ↩
                 order.";
            type yang:zero-based-counter64;
          }
          leaf lost {
            mandatory "true";
            description
              "Counter of lost packets.";
            type yang:zero-based-counter64;
          }
          leaf error {
            mandatory "true";
            description
              "Counter of packets with errors.";
            type yang:zero-based-counter64;
          }
          leaf max-delay {
            mandatory "true";
```

```
              description
                "Maximum packet delay.";
              type uint64;
              units "nanoseconds";
            }
            leaf min-delay {
              mandatory "true";
              description
                "Minimum packet delay.";
              type uint64;
              units "nanoseconds";
            }
            leaf avg-delay {
              mandatory "true";
              description
                "Average packet delay.";
              type uint64;
            }
            container packet-delays {
              description
                "Histogram of observed packet delays.";
              uses histogram64;
            }
          }
        }
        container blackhole {
          uses enabled-switch;
        }
        container physical-coding-sublayer {
          uses enabled-switch;
          leaf sys-loopback-enabled {
            description
              "Enable system loopback.";
            type boolean;
          }
          leaf net-loopback-enabled {
            description
              "Enable network loopback.";
            type boolean;
          }
          leaf low-power-enabled {
            description
              "Enable low power.";
            type boolean;
        }
        leaf rx-scr-enabled {
          description
            "Enable RX descrambler.";
          type boolean;
        }
        leaf tx-scr-enabled {
          description
            "Enable TX scrambler.";
          type boolean;
        }
        container data {
          config "false";
          leaf tx-fault {
            mandatory "true";
            description
              "TX fault indicator.";
            type boolean;
          }
          leaf rx-fault {
            mandatory "true";
            description
              "RX fault indicator.";
            type boolean;
          }
          leaf high-ber {
            mandatory "true";
            description
              "High BER indicator";
            type boolean;
          }
          leaf ber-count {
            mandatory "true";
            description
              "BER counter.";
            type uint8;
          }
          leaf errblk-count {
            mandatory "true";
            description
              "Counter of blocks with errors.";
            type uint8;
          }
          leaf block-lock {
            mandatory "true";
            description
              "Block lock indicator.";
            type boolean;
          }
        }
      }
    }
  }
  /* RPCs */
  rpc reset-modules {
    description
      "Reset (switch on/off) one or more COMET modules↪
          .";
    input {
      leaf-list submodule {
        description
          "Names of the modules.";
        type enumeration {
          enum input-packet-checker;
          enum output-packet-checker;
          enum sender;
        }
      }
    }
  }
  rpc reset-counters {
    description
      "Reset all counters in one or more COMET modules↪
          .";
    input {
      leaf-list submodule {
        description
          "Names of the modules.";
        type enumeration {
          enum statistics;
          enum input-packet-checker;
          enum output-packet-checker;
          enum physical-coding-sublayer;
        }
      }
    }
  }
  rpc send-traffic {
    description
      "Send traffic from active pcap file";
    input {
      leaf mode {
        description
          "Mode of sending";
        type enumeration {
          enum full-speed;
          enum user-speed;
          enum interval;
        }
      }
    }
  }
}
```

Listing D.1: COMET configuration data model

Appendix **E**

Contents of Enclosed CD

```
comet ........................ COMET control software library
  bin ........................ COMET binaries and packages
  sw ......................... Source code directory
    comstat .................. comstat(1) source codes
    libcomet ................. COMET library
    netcomet ................. Netopeer plug-in
      config ................. Netopeer plug-in configuration and installation
    pcapedit ................. pcapedit(1)
    pcap2sze ................. pcap2sze(1)
    yang ..................... COMET configuration data model
  images ..................... Sample images for comet.cgi
doc .......................... Documentation
  comet-doxygen .............. Documentation for COMET utilities
  libcomet-doxygen ........... Documentation for libcomet library
    html ..................... HTML version
    latex .................... PDF version
    man ...................... Manual pages
      man3 ................... Library functions man pages
fw ........................... COMET firmware
```

Figure E.1: Contents of Enclosed CD