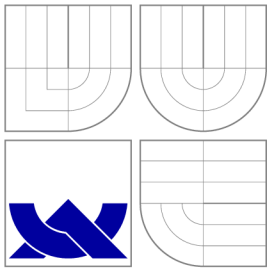
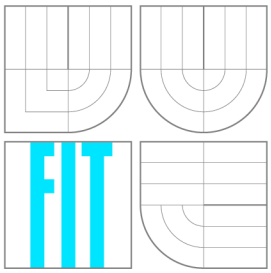


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

# SIMULACE A GENEROVÁNÍ SÍŤOVÉHO PROVOZU

NETWORK TRAFFIC SIMULATION AND GENERATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. JIŘÍ MATOUŠEK
AUTHOR

VEDOUCÍ PRÁCE — Ing. PAVOL KORČEK
SUPERVISOR

BRNO 2011

# Abstrakt

S rozvojem počítačových sítí umožnujících přenosy dat rychlostí 10 Gb/s a vyšší roste také potřeba vývoje nových síťových zařízení schopných pracovat na takovýchto rychlostech. Nově vyvinutá síťová zařízení je třeba před jejich nasazením do reálného provozu podrobit důkladnému testování, které se provádí pomocí přehrávání uměle vytvořeného nebo dříve zachyceného síťového provozu na lince vedoucí k testovanému zařízení, to vše také na maximální rychlosti linky. Současná testovací zařízení buď nejsou dostatečně výkonná, nebo kromě svého vysokého výkonu vynikají také vysokou cenou. Cílem této diplomové práce je tedy navrhnout hardwarově akcelerovanou aplikaci schopnou generování a přehrávání síťového provozu rychlostí 10 Gb/s. Pro akceleraci aplikace je použita karta COMBOv2 společně s platformou NetCOPE. Architektura navržené aplikace je modulární, což umožňuje s výhodou využít jednotlivé části aplikace pro implementaci různých funkcí. Mezi ty patří generování syntetického IPv4 nebo IPv6 provozu a dále pak přehrávání dříve zachyceného síťového provozu uloženého v paměti počítače. Generovaná i přehrávaná síťová data jsou vysílána do sítě rychlostí 10 Gb/s, přičemž na výstupu je možné omezit přenosovou rychlost až na hodnotu 1 Mb/s. V závěru práce je provedeno srovnání vlastností implementované aplikace s generátorem paketů implementovaným na platformě NetFPGA, které vyznívá příznivěji pro popisovanou aplikaci.



# Abstract

Development of computer networks able to operate at the speed of 10 Gb/s imposes new requirements on newly developed network devices and also on a process of their testing. Such devices are tested by replaying synthetic or previously captured network traffic on an input link of the tested device. We must be able to perform both tasks also at full wire speed. Current testing devices are either not able to operate at the speed of 10 Gb/s or they are too expensive. Therefore, the aim of this thesis is to design and implement a hardware accelerated application able to generate and replay network traffic at the speed of 10 Gb/s. The application is accelerated in the FPGA of the COMBOv2 card and it also utilizes the NetCOPE platform. Architecture of the application is modular, which allows easy implementation of different modes of operation. The application implements both capturing and replaying network traffic at full wire speed, but traffic can be limited to a specified value of bitrate at the output. The thesis is concluded by a comparison of the implemented application and the packet generator implemented on the NetFPGA platform. According to this comparison, the implemented application is better than the NetFPGA packet generator.



# Klíčová slova

generátor paketů, přehrávání síťového provozu, IPv6, NetCOPE, COMBOv2, pseudonáhodná čísla, 10 Gigabit Ethernet

# Keywords

packet generator, network traffic replaying, IPv6, NetCOPE, COMBOv2, pseudo-random numbers, 10 Gigabit Ethernet

# Citace

Jiří Matoušek: Network Traffic Simulation and Generation, diplomová práce, Brno, FIT VUT v Brně, 2011

# Network Traffic Simulation and Generation

## Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Pavola Korčeka. Další informace mi poskytli kolegové z projektu Liberouter. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jiří Matoušek
May 24, 2011

## Poděkování

Na tomto místě bych rád poděkoval panu Ing. Pavolu Korčekovi za jeho aktivní přístup k vedení této diplomové práce a rady, které mi při její tvorbě poskytnul. Mé poděkování patří také kolegům z projektu Liberouter, kteří mi pomáhali po stránce odborné i technické. Za výraznou podporu v mém snažení bych chtěl poděkovat také mé rodině.

# Contents

# Chapter 1

# Introduction

During the last two decades, we could register significant development of computer networks. This was caused mainly by the development of the Internet and by growing usage of its services. The most notable development we could see in the area of technologies for computer networks. Their improvements allowed increasing of data transfer rates. Nowadays, we are able — depending on used technology — to transfer data through a computer network at a speed of tens to hundreds Gb/s.

High-speed data transfers through a computer network imply increased requirements not only to transfer medium, but also to transmitting and receiving devices. If we consider the Ethernet technology, each two frames are separated by the time required for transmission of 96 b of data. This period is called the *Interframe Gap* (IFG) [12] and its value in computer networks supporting the *10 Gigabit Ethernet* (10GbE) standard is 9.6 ns. During this time, a transmitting device has to get ready for transmission of the next frame and a receiving device has to process an incoming frame and get ready for receiving the next frame. The IFG at the 10GbE is too short to allow processing network data using only a software application. To accomplish this task, we have to accelerate processing of network data in a hardware or to implement whole processing in hardware.

There are several possibilites of hardware acceleration of processing network data. One of these possibilities is represented by the *Field Pogrammable Gate Array* (FPGA) technology. The FPGA technology is used for hardware acceleration of processing network data e. g. at the NetFPGA project [3] or the Liberouter project [4]. As a part of the Liberouter project, the family of hardware acceleration cards called COMBOv2 [20] was developed and the NetCOPE [17] platform for this family of cards was designed and implemented. The main objective of this platform is to allow rapid development of hardware accelerated network applications able to transmit or receive network data at the speed of 1 Gb/s or 10 Gb/s.

Each newly developed network device requires extensive testing before its deployment into real operation. During this testing, we must ensure verification of the ability to transmit and/or receive network traffic at the maximum speed specified by the supported technology. If we develop devices for current backbone networks, we have to be able to generate or replay network traffic at the speed of 10 Gb/s. Sometimes we also require transmission of packets at exact time, which is useful for a simulation of different network topologies and behaviours.

The aim of this master's thesis is to design and implement a hardware accelerated network application able to generate synthetic network traffic and to replay previously captured real network traffic. The designed application must be able to perform both tasks

at the speed of 10 Gb/s. The application will be based on the COMBOv2 cards family utilizing the NetCOPE platform.

Content of this master's thesis is as follows. Chapter 2 contains a summary of already existing solutions, which could be used for generating or replaying network traffic. At the end of this chapter, there are highlighted positive aspects of the described solutions. Next chapter (3) presents all information necessary for designing a network aplication based on the NetCOPE platform. First of all, basics of the *Open Systems Interconnection* (OSI) Reference Model are mentioned. Next, there is a section about the COMBOv2 cards family and the chapter ends with description of the NetCOPE platform. The key part of the thesis is in chapter 4. This chapter describes an architecture of the designed application and presents details about each module of the design. Chapter 5 contains information about implementing the application presented in the previous chapter. The implemented application is evaluated in chapter 6. There is presented an overview of device utilization and a summary of the key features of the implemented application. Whole thesis is concluded in chapter 7, where achievements are summarized and possibilities for the further development are outlined.

Chapters 2, 3 and 4 are results of the work done in the term project. For the master's thesis, these chapters were slightly modified to fit in the structure of this thesis. Some sections of these chapters have also been newly added (e. g. the section about the OSI Reference Model).

# Chapter 2

# Related Work

For testing of network devices, we have to be able to generate synthetic network traffic or replay previously captured real network traffic on a link to a tested device. The testing is done by monitoring of the device behaviour under different circumstances implied by incoming network traffic. As a part of the testing, we also have to verify the ability to receive and process network data at full wire speed.

This chapter presents an overview of three different currently applied approaches to generating and/or replaying network traffic. The first approach is a pure software solution, while second one is pure hardware implementation. Third approach is based on an acceleration card with the FPGA chip. Using this overview, we will identify positive aspects of presented solutions, which we will keep in mind during designing of our application.

## 2.1 Software-Based Solutions

The most common way of generating network traffic is to use open source software tools for capturing packets and their subsequent replay (e. g. `tcpdump` [23] for capturing and `tcpreplay` [24] for replaying). The advantage of this approach is the use of ordinarily available software and hardware components. Unfortunately, generic hardware components can — depending on a vendor — vary in their behaviour and therefore the output traffic might be also very inaccurate as shown for example in [10]. Another disadvantage of this approach is also a very low throughput. For high-speed networks, it is impossible to use this approach for generating network traffic at full wire speed, mainly because of the operating system TCP/IP stack.

## 2.2 Hardware Network Testers

If we are looking for the state of the art devices for testing of network components, we will find commercial hardware network traffic generators playing this role. They are powerful devices for generating network traffic at full wire speed. These hardware generators provide broad options for setting the properties of generated traffic, so they are useful for many kinds of network tests and experiments. Their disadvantages include mainly very high price and also their proprietary nature makes them very inflexible for the research on new techniques and protocols.

As the representants of this category, we can mention systems from Spirent [7] or stochastically based generators from Ixia [13]. Ixia systems allow the users to create and

save synthetic traces to be rerun in the future. These systems can be useful, however, they do not allow replaying of previously captured real traffic. Also, it has been shown that properties of this kind of devices are not accurate enough for some experiments [6].

## 2.3 NetFPGA Packet Generator

To the best of my knowledge, there is only one FPGA-based network packet generator [8]. This *Stanford University Packet Generator* (SPG) utilizes so-called the NetFPGA platform, which consists of a hardware card with the Virtex-II Pro FPGA chip. The card is connected to a host computer using the PCI bus and we can find 4x1 Gb/s network interfaces and a quite small memory integrated on the card.

Packets to be sent by the SPG must first be loaded into the platform's memory from a PCAP file stored on the host, and only after that they can be transmitted to a network. This two-stage process means that the SPG can only replay short previously captured traces. The largest memory on the board is 64 MB which is about only 0.5 second of traffic at the speed of 1 Gb/s.

Using the on-board memory for buffering network traffic rises from the need of sending packets to the card over the PCI bus. There is a 33 MHz 32-bit bus on the NetFPGA platform with a theoretical top transfer rate of 1056 Mb/s, but there is a significant overhead even in the host where the bus is not shared. Most importantly, the number of *Direct Memory Access* (DMA) transfers between the driver and the platform is limited such that the total throughput is only 260 Mb/s when individually transferring 1518 B long packets. Some software improvements were made to the SPG to increase its ability to generate longer sequences at the speed of 1 Gbit/s [10]. However, these improvements caused sending only zero data in packets.

## 2.4 Summary of Positive Aspects

Identification of positive aspects of the existing solutions will be useful during specifying requirements on the designed application. It is almost for sure that we will not be able to keep all the positive aspects in our application (e. g. because some features are contradictory and it is necessary to find a reasonable trade-off among them), but their knowledge is necessary during designing the application. Keeping these features in mind will ensure utility of the application, because its properties will be at least comparable with the best-known network testing applications today.

We can summarize the main positive features of the presented solutions for generating or replaying network traffic into the following list (in order of clarification, each feature is followed by the corresponding above mentioned approach written in brackets)

- ease of use (*software-based solutions*)
- availability of required software and hardware (*software-based solutions*)
- ability to operate at full wire-speed (*hardware network testers*)
- great number of possible settings (*hardware network testers*)
- processing in a hardware at a reasonable price (*NetFPGA packet generator*)
- precise packets emission (*NetFPGA packet generator*)

# Chapter 3

# Prerequisites

Before we will deal with designing the application, we need to clarify details about the platform used for the development of our application and also about environment, where the application will be deployed and used. These information will be presented in following sections. First of all, the OSI Reference Model for network applications and devices is described. Next two sections contain information about the platform. Here we can find description of the COMBOv2 cards family and then the NetCOPE platform consisting of a software and a hardware part is presented.

## 3.1 OSI Reference Model

In the area of computer networks, we can find devices of many different vendors. Therefore, it is inconceivable to use a proprietary protocol for communication among a large group of network devices and we need international standards for such communication. There are standardization organizations developing and maintaining network standards. The most important of them are listed below.

- *International Organization for Standardization* (ISO)
- *International Telecommunication Union – Telecommunication Standardization Sector* (ITU-T)
- *Institute of Electrical and Electronics Engineers* (IEEE)
- *Internet Engineering Task Force* (IETF)
- etc.

A basis for development of communication protocols and network devices is represented by the OSI Reference Model [1], which has been created by the joint effort of the ISO and the ITU-T. This standard defines an abstract model of an open communicating system and also includes a specification of its features. It is not possible to implement a network system directly using this standard — the OSI Reference Model is rather intended to be a theoretical basis for defining other network communication standards and specifying newly developed network devices. In order to be able to serve as the reference model, the OSI Model has to be quite general. It is therefore quite common to use different simplifications of the OSI Reference Model — e. g. the Internet Protocol Suite [2] (also known as the

TCP/IP model). However, the OSI Model is irreplaceable in its position of the reference model.

The main idea brougth by the OSI Reference Model is a principle of layered architecture, where seven different layers are defined for the open communicating system (see Figure 3.1). For each layer (N) (except the Application Layer), there is defined one higher layer (N+1) and for each layer (N) (except the Physical Layer), there is defined one lower layer (N-1). The standard defines for each layer (N) a set of services, which this layer has to provide to the higher layer (N+1), however, implementation of these services is not specified. Request for the service of the lower layer and its granting to the higher layer are done using defined interface called *Service Access Point* (SAP).

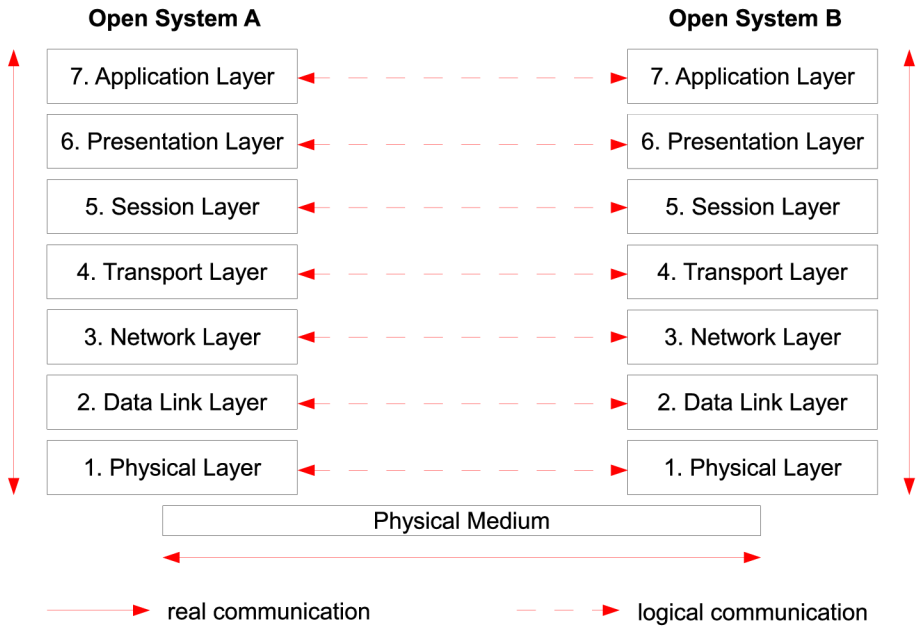



Figure 3.1: Layered Architecture of the OSI Reference Model

During communication of two different open systems, data are firstly transfered between layers of the source system using SAPs, then the data are transmitted to the destination system over physical medium and in the end, transfers between layers using SAPs take place at the destination system. Each layer of a system (except the lowest one and the highest one) can directly communicate only with the higher layer and the lower layer. Nevertheless, logical interconnections are established between corresponding layers of communicating systems. For ensuring these connections, lower layer services are used. Such a system of logical interconnections allows using a different communication protocol at each layer, so the way of communication and also data format can differ between the layers. In general, we refer to the layer-specific data format by term *Protocol Data Unit* (PDU). For PDUs at different layers we also use their common names shown together with a principle of data encapsulation in Figure 3.2.

Next two sections will describe details about two of the OSI Reference Model layers — the Network Layer and the Data Link Layer. From the point of view of the designed application, these two layers are the most important. The application itself will operate at the Network Layer and the NetCOPE platform implements the Data Link Layer.

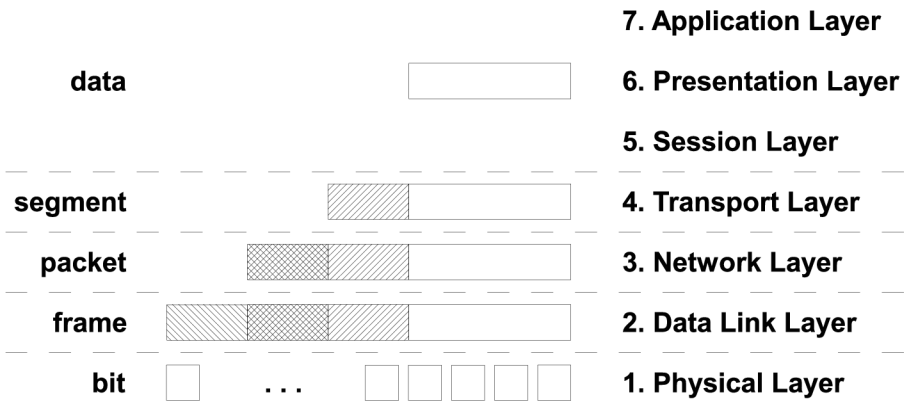



Figure 3.2: Layer-Specific PDU Names and the Data Encapsulation Principle

### 3.1.1 Network Layer

The main task of the Network Layer is to provide services for transparent network data trasmission over a computer network. Functions providing these services to the Transport Layer of the OSI Reference Model are in general described in the standard [1]. Further in this section, we will deal with the *Internet Protocol* (IP), which is the most common protocol of the Network Layer.

The IP was designed as a connection-less communication protocol for data transmissions over packet-switched networks, i. e. communication without explicit connection establishment. When the IP is in use, each packet can traverse a network over a different line, which implies possibility of changing the order of the packets at a receiving device. Since the IP does not deal with establishing the connection, its main purpose is to ensure addressing of packets (which also includes their routing through a network) and their possible fragmentation in order of their transmission over links with limited *maximum transmission unit* (MTU) size. To be able to provide these functions, the IP accompany transmitted data with control data forming an IP packet header.

The first widely deployed version of the IP was the *Internet Protocol version 4* (IPv4) defined in *Request for Comment* (RFC) 791 [27]. This document also defines the IPv4 header format, which is shown in Figure 3.3. Fields of the IPv4 header are described in the following definition list.

**Version (4 b)** — version of the IP (value 4 indicates the IPv4).

**IHL (4 b)** — *Internet Header Length* in 32-bit words (minimum value for a correct header is 5)

**Type of Service (8 b)** — this field contains indication of a service transported by the packet. In some networks, this can be used for providing the quality of service control. Two least significant bits are reserved for future use (and always set to 0). Use of other bits is explained in [27].

**Total Length (16 b)** — value in this field represents a length of the whole packet (header + data) in octets. The maximum packet length is 65 535 octets, however, according to [27], only packets of total length less than or equal to 576 octets are recommended.

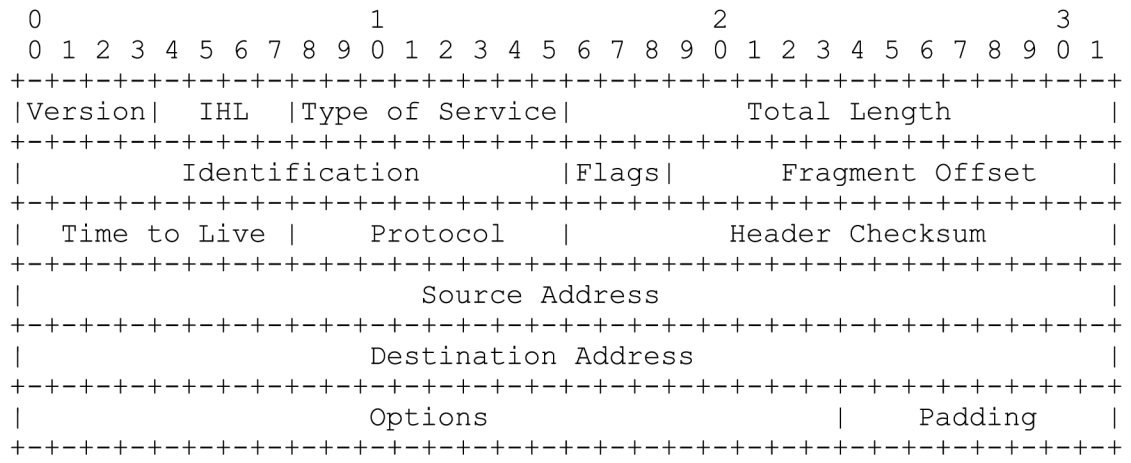



Figure 3.3: Structure of the IPv4 Header [27]

**Identification (16 b)** — value provided by a sender in order to help with reassembling of a fragmented packet.

**Flags (3 b)** — the most significant bit is reserved (must be 0) and next two bits contain two flags for fragmentation control.

**Fragment Offset (13 b)** — value of an offset of a fragment from the begining of an original packet. The offset is measured in units of eight octets (64 b).

**Time to Live (8 b)** — the maximum time the packet can traverse a network. Time to live is measured in seconds, but during each processing of the header, this field must be decreased by at least 1.

**Protocol (8 b)** — identification of an upper layer protocol according to [21].

**Header Checksum (16 b)** — checksum of IPv4 header fields computed by an algorithm specified in [27]. This value is verified and recomputed each time the header is processed by a network device (e. g. because changes in the Time to Live field).

**Source Address (32 b)** — an IPv4 address of the source device.

**Destination Address (32 b)** — an IPv4 address of the destination device.

**Options (varible length)** — a place for adding further information to the packet. This field may appear or not. The Options field format is specified in [27].

**Padding (variable length)** — field of a variable length with each bit set to 0 and ensuring that the IPv4 header ends on the 32-bit boundary.

Due to the insufficient address range of the IPv4, the *Internet Protocol version 6* (IPv6), described in RFC 2460 [9], was presented in 1998. While the IPv4 addresing scheme allows $2^{32}$ different addresses, in the IPv6 there is possible to assign $2^{128}$ of different addresses. The IPv6 also tries to simplify packets processing on routers, so an IPv6 header is of a fixed size (40 B) and less important header fields were moved to so-called extension headers.

Faster routing process is also supported by omission of a header checksum control and by forbidding packets fragmentation on routers. Format of the IPv6 header is shown in Figure 3.4, which is followed by description of header fields.

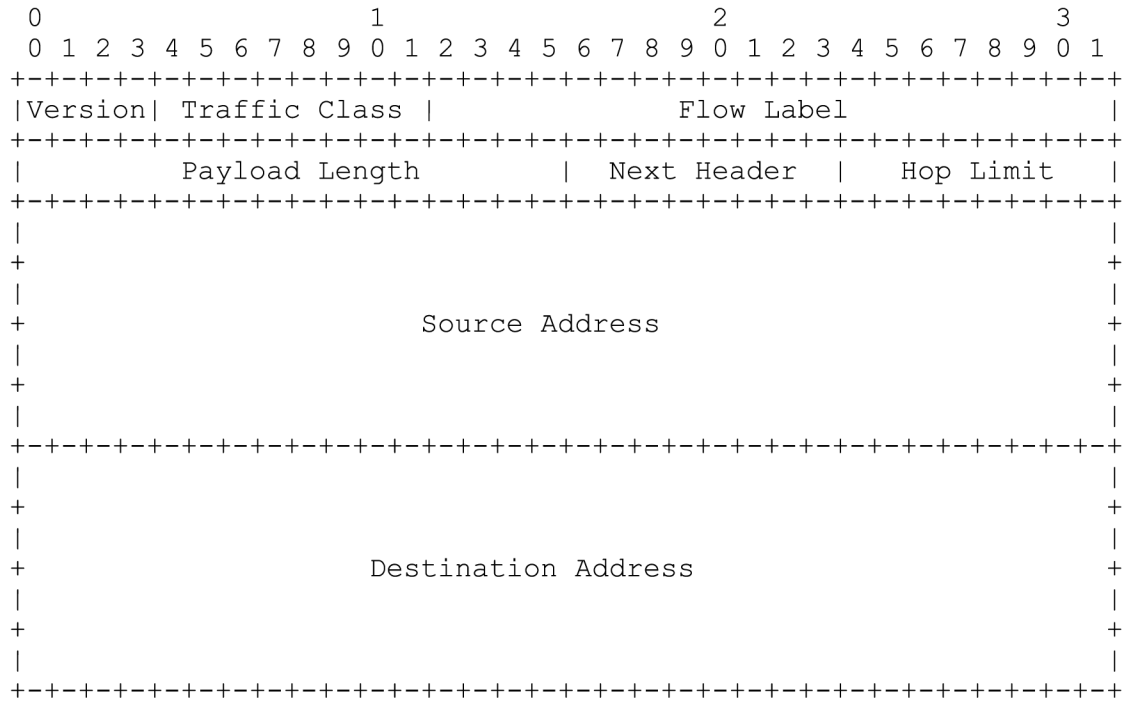



Figure 3.4: Example IPv6 Header [9]

**Version (4 b)** — verison of the IP (value 6 indicates the IPv6).

**Traffic Class (8 b)** — indentification of a traffic class, which can be used (like the Type of Service field in the IPv4 header) for the quality of service control.

**Flow Label (20 b)** — identification of a flow of IPv6 packets, which can be seen as a way of grouping packets of the same properties.

**Payload Length (16 b)** — unsigned integer value determining number of octets in the packet payload (i. e. between end of this header and the end of the packet).

**Next Header (8 b)** — this field identifies a type of an extension header immediately following the IPv6 header. Identifiers are the same as for the Protocol field of the IPv4 header (see [22]).

**Hop Limit (8 b)** — the maximum number of forwarding, which can the packet undergo. Each forwarding decrements the value in the Hop Limit field by one and the packet is discarded when the value reaches zero.

**Source Address (128 b)** — an IPv6 address of the source device.

**Destination Address (128 b)** — an IPv6 address of the destination device.

### 3.1.2 Data Link Layer

The Data Link Layer of the OSI Reference Model is expected to provide services required by the Network Layer for accomplishing its tasks. Another objective of the Data Link Layer is to detect and try to correct errors, which can possibly occur at the Physical Layer. Probably the best-known Data Link Layer implementation is the Ethernet technology designed for use in *Local Area Networks* (LAN) and defined in a group of the IEEE standards 802.3 [12].

In order to ensure functionality defined for the Data Link Layer, the standard defines the Ethernet frame structure containing control data along with transferred data of the Network Layer. Structure of the Eternet frame is shown in Figure 3.5. The maximum frame size (including all fields from the Destination Address to the Frame Check Sequence) is defined to be 1518 B. Frame transmission is always preceded by transmitting the Preamble and the *Start Frame Delimiter* (SFD). After transmission of a frame, the next frame cannot be transmitted earlier than after time equivalent to transmission of 96 b at the maximum speed of a link. This time interval is called *Interframe Gap* (IFG) and its size is for example 96 ns when the *Gigabit Ethernet* (GbE) is in use or 9.6 ns in case of the 10GbE. The following list contains the description of particular Ethernet frame fields.

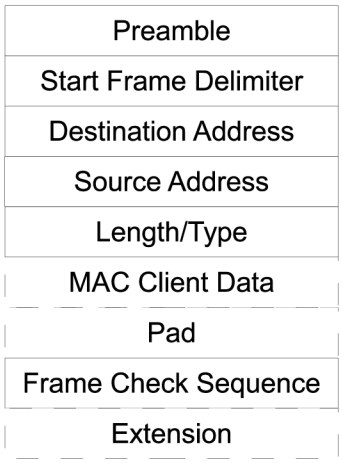



Figure 3.5: Ethernet Frame Structure

**Preamble (7 B)** — this field consists of seven octets, each containing the data pattern `10101010`. Its main purpose is to allow synchronization between a sending device and a receiving device.

**Start Frame Delimiter (1 B)** — the last octet before the frame. It contains the bit sequence `10101011`.

**Destination Address (6 B)** — a *Media Access Control* (MAC) address of the destination device. It can be an individual (unicast) address or a group (multicast or broadcast) address.

**Source Address (6 B)** — a MAC address of the source device. It is always an individual (unicast) address.

**Lenght/Type (2 B)** — in case of the value less than or equal to 1 500, it determines the number of octets in the MAC Client Data field. If the value of the Length/Type

field is greater than or equal to 1 536, then it specifies a protocol of the higher layer encapsulated in the MAC Client Data field.

**MAC Client Data + Pad (46 – 1 500 B)** — the field MAC Client Data contains data provided by the higher layer. This field can be of the maximum size of 1 500 B. If it is shorter than 46 B, the Pad field fills remaining octets.

**Frame Check Sequence (4 B)** — value in this field is a 32-bit *cyclic redundancy check* (CRC) computed from the content of the fields Destination Address, Source Address, Length/Type, MAC Client Data and Pad.

**Extension (variable length)** — under special conditions defined in the standard [12], this field can extend the Ethernet frame.

## 3.2 Family of COMBOv2 Cards

The family of COMBO cards was designed by CESNET [29] and dedicated for building hardware accelerated network applications. The main idea of this family of cards is the maximum flexibility provided both by the mother card extendable by a set of add-on cards and by the FPGA chip on the mother card. Thanks to using add-on cards, it is possible e. g. to change a number of network interfaces or to connect an external precise clock signal generator. On the other hand, the FPGA chip allows changing of COMBO card's functionality while maintaining benefits of hardware acceleration.

Currently, the second generation of the COMBO cards family (COMBOv2) is in use [30]. Figure 3.6 shows the COMBOv2 mother card with attached add-on interface card containing two 10 Gb/s interfaces. Details about the mother card and the set of add-on cards will be presented in following sections.

Figure 3.6: COMBOv2 Mother Card with Attached 2x10 Gbit/s Interface Card [31]

### 3.2.1 Mothercard

The COMBOv2 mothercard is a foundation of the whole family of cards. This hardware acceleration card provides high computing power especially for network applicatons. Thanks to the FPGA chip, processing of network data from 10GbE links can be done in parallel. Pre-processed data are transfered to the host *Random Access Memory* (RAM) through the PCI Express interface. The mothercard also contains a static and a dynamic memory for storing already processed data or data for further processing. A unique boot system utilizing smaller FPGA chip allows on the fly card reboot. Summary of basic mothercard parameters taken from [30] follows.

- Virtex-5 (XC5VLX155T) FPGA
- PCI Express x8 interface
- 2x QDR II RAM
- SODIMM DDR2 connector for up to 2 GB memory module
- 4x *Low Speed Connector* (LSC) with throughput up to 8 Gb/s
- 2x *InterFace Connector* (IFC) with bidirectional throughput up to 28 Gb/s
- Spartan-3E (XC3S1200E) FPGA controlling on the fly Virtex-5 reboot

### 3.2.2 Add-on Cards

While the mothercard provides mainly enough computing power for demanding network applications, the set of add-on cards ensures flexibility of the whole COMBOv2 cards family. Cards from this set usually do not contain the FPGA chip and they are provided with a fixed configuration. The most important subset of the set of add-on cards consists of interface add-on cards. Following list presents different configurations of interface add-on cards as well as other add-on cards.

- COMBOI-10G2 — the interface card with two 10 Gb/s interfaces
- COMBOI-1G4 — the interface card with four 1 Gb/s interfaces
- COMBOL-GPS — add-on card for receiving the GPS signal in order to provide a precise *clock* (CLK) and a *pulse per second* (PPS) signal
- etc.

## 3.3 NetCOPE Platform

For rapid development of hardware accelerated network applications on the COMBOv2 cards, the NetCOPE platform [17] has been designed. This platform consists of the firmware for the FPGA chip on the mother card and the software for accessing the card's functionality from an operating system. These two layers jointly handle connection of the mother card to the PCI Express and fast DMA transfers between the host RAM and this card. The software and the firmware part of the NetCOPE platform are described in respective sections followed by the section summarizing results of DMA transfers test performed on the NetCOPE platform.

### 3.3.1 Software

The NetCOPE platform software layer handles control of the COMBOv2 cards and allows the user to use functionality implemented using this platform in the FPGA. The software layer of the NetCOPE platform is built hierarchically and it can be divided horizontally or vertically [25].

In a horizontal manner, we can divide the platform software into three levels — drivers, libraries and software tools. The system drivers are used for the COMBOv2 cards control and they provide a basic interface between the hardware and the software. Above system drivers, there is a set of software libraries implementing functions based on operations provided by the drivers. These functions are used for implementing the software tools deployed to work with the card and the platform.

Platform software can be also divided vertically into two logical groups. The first group covers input/output operations with the COMBOv2 card, card initialization, the boot process for loading a firmware to the FPGA chip, monitoring and configuration of hardware components. The second logical group provides software support for the fast DMA transfers of user specific data to user applications.

Software provided together with the NetCOPE platform can be used to control any user application without any futher changes.

### 3.3.2 Firmware

Firmware of the NetCOPE platform comprises three main components — the *Generic Interconnection System* (GICS) [19], the DMA Module and the Network Module [18]. The user application is tied with all three modules as shown in Figure. 3.7. The main modules of the NetCOPE platform are highlighted in the figure by a blue colour and the user application is yellow.

The first main component of the NetCOPE platform is the GICS. It provides connection of all other NetCOPE and application components to the system bus through an interconnection system on the FPGA chip. This interconnection system is called the Internal Bus and it is organized in a tree topology. Data width of the Internal Bus is parametric and it can be chosen in the range from 8 to 128 b. The whole interconnection system is built using only four different components — the Root, the Switch, the Transformer and the Endpoint. The Root is the main component of the interconnection system and it controls connection to the system bus. On the COMBOv2 cards, the Root is the PCI Express endpoint. The Switch and the Transformer components are used to create the GICS tree structure. The Switch divides the Internal Bus into two new branches, each with different address range. The Transformer ensures transformation of bus signals between two parts of a different data width. The last component of the GICS is the Endpoint. This component terminates each GICS branch and it provides an interface for connection of components outside the GICS.

Second main part of the NetCOPE platform firmware layer is the DMA Module. This component takes care about fast DMA transfers between the host memory and the FPGA chip and it can be divided into three submodules — the Buffer separate for each *receiving* (RX) and *transmitting* (TX) channel, the Driver and the Descriptor Manager. In the RX direction, data from a hardware accelerated application are received and stored in the Buffer, which is considered to be circular. When new data are present in the Buffer, the Driver obtains descriptors of a circular buffer in the host memory from the Descriptor Manager and after that, the Driver initializes a DMA transfer from the FPGA chip to the host memory. Data are sent off to the host memory via the Internal Bus interface.

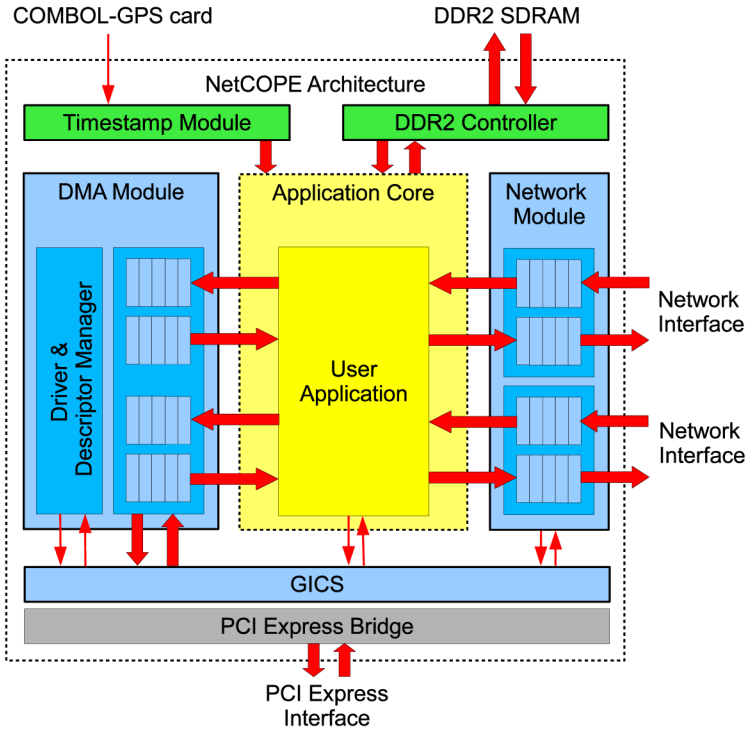



Figure 3.7: Firmware of the NetCOPE Platform

Realization of DMA transfers in the TX direction is similar to DMA transfers in the RX direction and it also uses the Buffer, the Driver and the Descriptor Manager.

The last main component of the platform is the Network Module. The aim of deploying this module is to provide simplified access to network interfaces. For each RX and TX direction of each network interface, there is one buffer — *input buffer* (IBUF) in the RX direction or *output buffer* (OBUF) in the TX direction. These buffers are transforming data between the FrameLink protocol [26] at the user application side and the GMII or the XGMII protocol at the network side. Beside this, both the IBUF and the OBUF incorporate different counters of processed frames (e. g. total processed frames, correct frames and discarded frames). Moreover, in the IBUF, numerous statistics are calculated for each incoming frame and frames can be discarded based on these statistics.

So far described modules of the NetCOPE platform are the most important ones. They are also sufficient for building a basic network interface card. However, the platform contains also some expansion modules, higlighted by a green colour in Figure 3.7. In this group, we can include the Timestamp Module, which can optionally provide very precise 64 b timestamps. This module uses a PPS signal from the COMBOL-GPS card (if present) or an internal PPS signal. Based on the selected PPS signal, timestamps with nanosecond resolution are generated and provided to other components implemented in the FPGA chip. Another example of en expansion module is the *Double Data Rate* (DDR2) Controller, which allows read and write operations on a dynamic memory in the *Small Outline Dual In-line Memory Module* (SODIMM) DDR2 connector.

### 3.3.3 Platform Performance

Since related documents about the COMBOv2 cards family and the NetCOPE platform (e. g. [17] and [20]) describe only high performance operation in the RX direction, we had to examine its performance in the TX direction, mainly between the host and the card. The reachable throughput from the RAM on host computer to the COMBOv2 internal memory has been tested using DMA transfers over the PCI Express x8 link. Two different configurations of the host computer were chosen, as shown in Table 3.1. The first one is a standard server platform and the second is a powerful gaming platform.

| **Parameter** | **1st configuration** | **2nd configuration** |
|---|---|---|
| CPU | 2 x Xeon(R) CPU E5420 @ 2.50 GHz | Intel(R) Core(TM) i7 CPU 920 @ 2.67 GHz |
| Mainboard | Supermicro X7DB8 | Supermicro X8STE |
| Chipset | Intel®5000P | Intel®X58 |

Table 3.1: Configuration of Tested Host Computers

All other parameters not shown in Table 3.1 were kept unchanged. It is especially the host RAM memory (4 GB) and the operating system version (32-bit Linux-2.6.26.3). DMA engine runs in our test design at 218.75 MHz, and one data transfer bulk consists of $20 \cdot 10^3$ transfers. Each test was performed with a different data block size from 4 B up to the PCI Express maximum of 4 kB (4 096 B). Measurement was performed by utilizing a FPGA internal counter, which started to count before and stopped immediately after all the transfers were done. Results of tests are shown in Figure 3.8. It clearly demonstrates that two different configurations reached highly different results. By this test, it was shown that it is not possible to reliably generate packets from software on an arbitrary server platform. For higher packet rates it is necessary to use the DDR2 memory on the COMBOv2 card for storing packets to be later sent to a network (capture & replay scenario). Another possibility, if there is no sufficient platform, is generating packets synthetically without fruitless loading of the PCI Express.

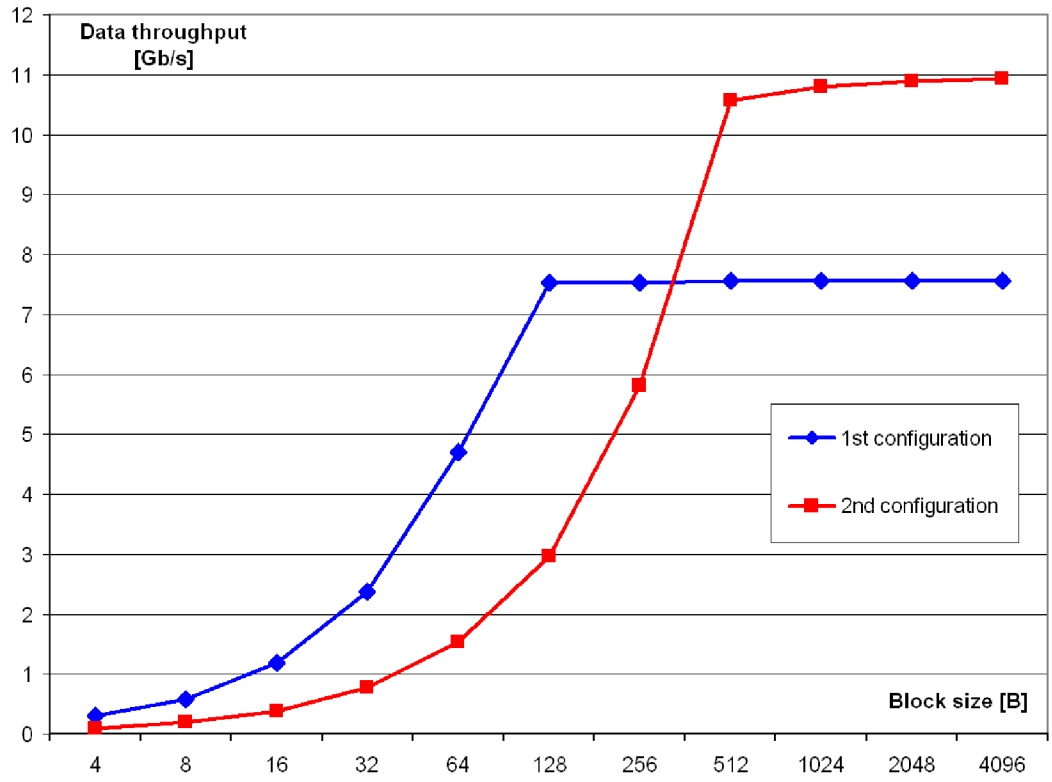



Figure 3.8: RAM to FPGA Throughput — FPGA Design Working Frequency: 218.75 MHz, MaxReadRequestSize (PCI Express Parameter): 2 048 B

# Chapter 4

# Design

This chapter represents the core of the master's thesis. The main aim of the theoretical work on this thesis is to design the hardware accelerated network application utilizing the NetCOPE platform. This applicaton should be able to generate synthetic network traffic and also to replay previously captured real network traffic at the speed of 10 Gb/s. Architecture of the application is discussed in the following sections.

During the design phase, we will need to use the knowledge presented in chapters 2 and 3. Based on the functionality provided by the COMBOv2 family of cards and by the NetCOPE platform, we can sketch the outlines of the designed application. The knowledge of the Internet Protocol and the Ethernet technology allows us to fine-tune details of the design. If we compare this draft of the application with positive aspects of other similar applications, we get a final specification of the designed application, which can be summarized as follows

- ability to generate synthetic network traffic
- ability to replay previously captured real network traffic
- possibility to use common software tools for capturing and replaying network traffic
- ability to operate at full wire speed (i. e. at the speed of 10 Gb/s)
- support of the IPv4 and the IPv6
- possibility to limit network data transmission
- ability of precise packets emission

## 4.1 Packet Generator

Although generating packets is just one of many proposed functions of the application, it is considered to be the main function. Because of its importance, the whole application is sometimes reffered to as the packet generator.

Proposed packet generator will be based on the NetCOPE platform and it will fully use the functionality provided by the platform modules. The generator is designed to be a user application for the NetCOPE platform and it consists of newly implemented modules shown in Figure 4.1. This is the architecture for one network interface. In designs with multiple network interfaces, the basic architecture will be repeated for each network interface and some other minor changes will be made.

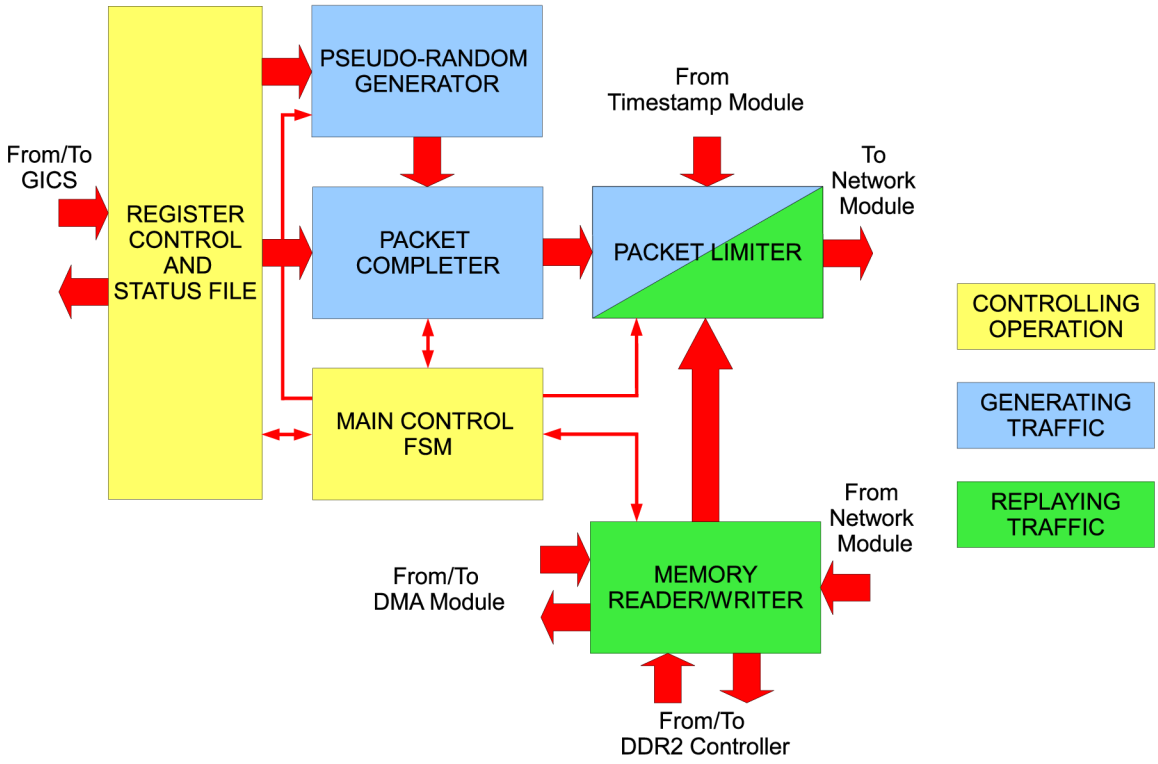



Figure 4.1: Modular Architecture of the Proposed Application for One Network Interface (Distinct Colours Show Usage of Submodules for Providing Different Functionality)

## 4.2 Modes of Operation

It is clear, that all functions of the proposed packet generator — as they were described earlier in this chapter — cannot be active at the same time. Therefore, the application will be able to operate in distinct modes, providing different functionality in each of them. In total, there will be four possible modes of operation. Switching the modes will be done by writing an appropriate 2-bit value into a control register of the generator. Value of these two bits will be reflected by the Main Control FSM, which will ensure correct operation of other submodules. Each of four different modes of operation is described in the further text.

### 4.2.1 Generating Synthetic Network Traffic

This mode of operation provides the main function of the application — generating synthetic IPv4/IPv6 network traffic at wire speed. Content of particular IPv4 or IPv6 header fileds is generated using a pseudo-random number generator. After that, the packet header is assebled from the generated values and a whole packet (including an attached payload) is transmitted to a network.

### 4.2.2 Loading Data to Dynamic Memory

Before it is possible to transmit previously captured real network traffic, network data have to be loaded into the dynamic memory on the COMBOv2 mothercard. This task

is accomplished in the second mode of operation. Content of the dynamic memory can be loaded either from a PCAP file stored in the host memory or directly from a network interface.

### 4.2.3 Transmission of Data From Dynamic Memory

If the previous mode of operation was used to load data into the memory, this mode ensures transmission of stored network traffic to the network. Based on an attached interface add-on card, the transmission can be done at the speed of 1 Gb/s or 10 Gb/s. Transmission of data can be limited to a specified bitrate or the limitation process can be based on precise 64-bit timestamps (if they were provided together with data).

### 4.2.4 Standard NIC Operation

When this mode of operation is set, the application works as a standard network interface card. Particular submodules of the design operate in this mode as less as possible or even do not operate at all. Therefore, we can describe this situation as that the application is not in operation.

In this mode of operation, it is possible to use common software tools for capturing and replaying network traffic (`tcpdump`, `tcpreplay`). As we have shown in section 3.3.3, on some platforms it is possible to transfer data from the host memory to the COMBOv2 card with a bit rate higher than 10 Gb/s. Thanks to this, we could transmit data via one 10 Gb/s network interface at full wire speed. Unfortunately, this mode of operation cannot guarantee wire speed for both 10 Gb/s interfaces of the add-on card. To overcome this, the dynamic memory must be used as described above.

## 4.3 Generator Submodules

The modular architecture of the generator shown in Figure 4.1 and discussed in details in this section allows less complex design phase of the whole application. Thanks to division the architecture into submodules, the design of the application is straightforward and easily understandable. Moreover, it reduces complexity in designing particular submodules and allows their faster and more comfortable development (e. g. by possibility to test separately each submodule). From the functional point of view, it is convenient that different submodules can be activated or deactivated in distinct modes of operation presented in section 4.2.

### 4.3.1 Pseudo-Random Generator

This section is divided into two parts. Firstly, different hardware implementations of a pseudo-random number generator are discussed and compared. There are also few words about transformation of generated sequences of numbers into the predefined range. Secondly, this section specifies reqiurements on pseudo-random number generators imposed by their application in generating synthetic IP traffic.

#### Generating Pseudo-Random Numbers in Hardware

The simplest to implement hardware pseudo-random number generator is based on *linear feedback shift register* (LFSR), which can also effectively be implemented in FPGAs [5].

This type of generators are based on a shift register with a feedback function. The input of the feedback function is driven by two or more bits from the register. Bit positions are given by the primitive polynomial coefficients, also called taps. The computed output value is fed to the input of the whole shift register (Fibonacci scheme) or alternatively to the input of the next register (Galois scheme). For more details see [15]. As the feedback function, exclusive-OR or its negation is often chosen.

LFSRs are easily implementable in the FPGA, but, on the other hand, a generated sequence of numbers has not very good properties, which can be shown e. g. by the so-called serial test [14]. To overcome this issue, a new architecture of *multiple LFSR used in parallel* (MLFSR) was proposed and firstly introduced in [15]. This improved version of LFSR can successfully pass even the Diehard battery test of randomness [16]. Table 4.1 shows effective score from the Diehard test of simple LFSR, MLFSR and also a true random generator, where the randomness comes from an atmospheric noise [11]. Because of these very good properties, MLFSR will be used as the pseudo-random number generator in the proposed application.

| **Generator type** | **Effective score** |
|---|---|
| true | 22 |
| MLFSR | 154 |
| LFSR | 756 |

Table 4.1: Score from the Diehard Test (Lower is Better)

In our application, we will often require generating values from a predefined interval (typically for IP addresses). For this purpose, equation 4.1 will be used for number conversion to the desired interval.

$$X = a + \frac{N \cdot (b - a)}{N_{max}} \tag{4.1}$$

The pseudo-random number generator output $N$ is in the range $(0, N_{max}>$ and $X$ is a new transformed number in the <a,b> range. For fast FPGA implementation, division by the $N_{max}$ value will be done by shifting. For multiplication, the application will utilize hard DSP48 slices available in Virtex-5 FPGAs used on the COMBOv2 cards. It is also important to note, that MLFSR is capable of generating zero values. This could not be normally reached with standard LFSR (not allowed state).

### Generating Synthetic IP Packets

Generating values of IP header fields can be done in three different modes

1. constant value of the field is specified by the user
2. the field is filled by the next value from a sequence of numbers
3. value for the field is randomly generated from the specified range

In order to ensure all three modes of generating header field values, we have to store information about the currently selected mode (separately for each header field) and also some special values requested in the selected mode. In the first mode, we have to store

the value of the field specified by the user. The second and the third mode share values of boundaries of the numeric sequence, but in the second generating mode we also have to specify a step size.

Before we will describe details about the way of generating particular IPv4 and IPv6 header fields, it has to be mentioned that the proposed application will not allow generating the Options filed of the IPv4 header and also generating of IPv6 extension headers will not be supported. This restriction means, among others, that a generated IP header will always be of a constant size (20 B in case of the IPv4 and 40 B in case of the IPv6). The last restriction of the generating process will take place during generating the IPv4 header, when generating fragmented packets will not be allowed.

Tables 4.2 and 4.3 present an overview of possibilities in generating distinct IPv4 and IPv6 header fields.

| **Name** | **Size (bits)** | **Position** | **Generated or Constant** | **Value (decimal)** |
|---|---|---|---|---|
| Version | 4 | 0 to 3 | constant | 4 |
| IHL | 4 | 0 to 3 | constant | 5 |
| Type of Service | 8 | 0 to 5 | generated | 0 to $2^6 - 1$ |
| | | 6 to 7 | constant | 0 |
| Total Length | 16 | 0 to 15 | generated | 21 to 1 500 |
| Identification | 16 | 0 to 15 | constant | 0 |
| Flags | 3 | 0 | constant | 0 |
| | | 1 | generated | 0 to 1 |
| | | 2 | constant | 0 |
| Fragment Offset | 13 | 0 to 12 | constant | 0 |
| Time to Live | 8 | 0 to 7 | generated | 1 to $2^8 - 1$ |
| Protocol | 8 | 0 to 7 | generated | 0 to $2^8 - 1$ |
| Header Checksum | 16 | 0 to 15 | constant | 0 |
| Source Address | 32 | 0 to 32 | generated | 0 to $2^{32} - 1$ |
| Destination Address | 32 | 0 to 32 | generated | 0 to $2^{32} - 1$ |

Table 4.2: Overview of Generating IPv4 Header Fields

Since some parts of fields Type of Service and Flags can be generated and some parts cannot, there is the column Position for specifying a bit range in the field, for which values in the last two columns of the same row hold. Another surprising information can be found in the row designated for the field Header Checksum, which is considered to be constant. The value of this fild will be computed in the submodule Packet Completer and an algorithm for its computation requires an initial value of this field to be zero [27].

If we look in the column Value, the range specified for the Total Length field may look strange. This value is based on following considerations. A payload of a packet should be of a length at least 1 B (which implies the total length of 21 B) and the length of the packet should not exceed the maximum allowed size of higher layer data in the Ethernet frame (which is 1 500 B — see section 3.1.2). It is also important to note, that the minimum allowed value for the field Time to Live is one, because the value zero would not make sense.

| Name | Size (bits) | Generated or Constant | Value (decimal) |
|---|---|---|---|
| Version | 4 | constant | 6 |
| Traffic Class | 8 | generated | 0 to $(2^8 - 1)$ |
| Flow Label | 20 | generated | 0 to $(2^{20} - 1)$ |
| Payload Length | 16 | generated | 1 to 1 460 |
| Next Header | 8 | constant | 0 |
| Hop Limit | 8 | generated | 1 to $(2^8 - 1)$ |
| Source Address | 128 | generated | 0 to $(2^{128} - 1)$ |
| Destination Address | 128 | generated | 0 to $(2^{128} - 1)$ |

Table 4.3: Overview of Generating IPv6 Header Fields

Similarly to generating fields of the IPv4 header, in Table 4.3 we can see extraordinary allowed values at rows designated for fields Payload Length and Hop Limit. In case of the field Payload Length, we can use the same reasoning for deriving the range of allowed values as for the field Total Length of the IPv4 header. Different final range is caused by different semantics of this field in the IPv6 header, where this field specifies a length of data without the header of the size 40 B. The range of allowed values for the field Hop Limit is absolutely the same as for the field Time to Live of the IPv4 header.

Generating the IPv4 or IPv6 header is initiated by the Packet Completer submodule, which sets a request signal. Based on the active value of this signal, the Pseudo-Random Generator starts generating IP header fields. When the generating process is complete, the Pseudo-Random Generator pass data to the Packet Copleter together with an active valid signal. The whole packet header is transferred at once and the further processing is left to the Packet Completer.

### 4.3.2 Packet Completer

This module of the generator ensures forming of an appropriate IP header from values generated in the Pseudo-Random Generator. These values represent content of the IPv4 or IPv6 header fields shown in Figure 3.3 or 3.4. The mechanism of passing data between the Pseudo-Random Generator and the Packet Completer has been described at the end of the previous section.

The first operation on generated data is a computation of header fields checksum. Since the field Header Checksum is a part of only the IPv4 header, this computation is done only when the IPv4 header was generated. The algorithm for computing the header checksum is described in [27] using following two sentences.

> *The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words in the header. For purposes of computing the checksum, the value of the checksum field is zero.*

For the next task of the Packet Completer, a value of the Total Length field (in case of generating the IPv4 header) or the Payload Length (when the IPv6 header was generated) is very important. This field determines the number of octets of a payload, which will be appended to the already formed header in order to form the whole packet. The payload is filled by a 1 byte long data pattern specified by the user in a corresponding register.

Before transmission of the packet, it has to be encapsulated in an Ethernet frame. The packet is preceded by he Ethernet frame fields Destination Address, Source Address and Length/Type. A content of the address fields is specified by the user by writing selected values into specified registers. The Length/Type field is filled by a value derived from the knowledge of the total length (IPv4) or the payload length (IPv6).

In the end, the complete IP packet extended by three fields of the encapsulating Ethernet frame is sent to the Packet Limiter, which takes care about its transmission to a network through an output network buffer. The transmission between the Packet completer and the Packet Limiter is done using the FrameLink protocol.

### 4.3.3 Memory Reader/Writer

This module of the generator is dedicated to take care mainly about communication with the dynamic memory on the COMBOv2 mother card using the DDR2 Controller module of the NetCOPE platform. The memory is used for storing real network traffic and therefore communication with the memory takes place in two modes of operation of the application. Another task of this part is to ensure a correct interconnection between three different types of data interface — a Network Module interface, a DMA Module interface and a DDR2 Controller interface — all of them full-duplex (see Figure 4.2).

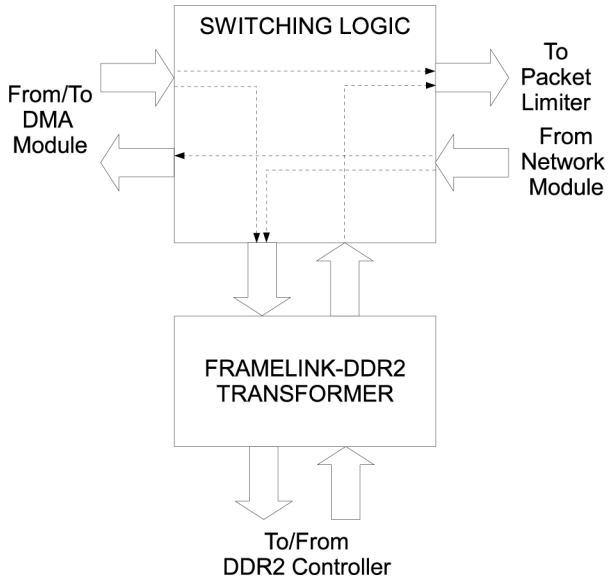



Figure 4.2: Internal structure of the Memory Reader/Writer module

The Switching Logic is the part providing the possibility of a flexible interconnection reflecting requirements imposed by the modes of operation specified in section 4.2. An input part of the DDR2 Controller interface can be connected to both the DMA Module and the Network Module, because storing real network traffic to the memory is possible from the host memory as well as directly from the network interface. An output of the DDR2 Controller is always connected to the Network Module interface through the Packet Limiter in order to transmit stored real network traffic to the network. When the generator is not in operation, the DMA Module interface is connected directly to the Network Module interface.

Between the Switching Logic and the DDR2 Controller interface, there is a component transforming data from the FrameLink protocol to a protocol used at a user interface of the DDR2 Controller. The FrameLink protocol was designed as a part of the NetCOPE platform and its detailed description is available in [26]. In order to design the FrameLink-DDR2 Transformer we will need to shortly remind signals of the FrameLink protocol (see Table 4.4), because their simplified version will be stored into the dynamic memory together with network data. Description of the DDR2 Controller user interface as well as the whole controller can be found in [28].

| **Signal** | **Width (bits)** | **Direction** |
|---|---|---|
| CLK | 1 | input |
| DATA | 64 | source → destination |
| SOF_N | 1 | source → destination |
| EOF_N | 1 | source → destination |
| SOP_N | 1 | source → destination |
| EOP_N | 1 | source → destination |
| REM | 3 | source → destination |
| SRC_RDY_N | 1 | source → destination |
| DST_RDY_N | 1 | destination → source |

Table 4.4: Overview of FrameLink Signals

The use of the DDR2 Controller imposes some requirements on the FrameLink-DDR2 Transformer and we will need to keep these requirements in mind during designing the transformer. Two main restrictions are a requirement on writing two 64 b blocks of data in one clock cycle and performing write operations in bursts of the size of four data blocks, i. e. writing 256 b of data in two successive clock cycles. Since a data width of the FrameLink protocol will be set to 64 b, it will be possible, under usual circumstances, to initiate the write operation to the dynamic memory only once in four clock cycles.

Based on the above mentioned facts about the FrameLink protocol and on the restrictions implied by the use of the DDR2 Controller, it seems to be the best solution to initiate write operation on the user interface of the DDR2 Controller not earlier than four 64 b data blocks will be available. To sake of simplicity of the transformer, FrameLink data will be stored in the dynamic memory together with the simplified version of the FrameLink protocol transporting these data. Therefore, in the memory we will recognize three types of blocks, each of the size of 64 b.

- data block
- control block
- padding block

The data block contains data transported by the FrameLink protocol and the Control block is dedicated for storing important values of FrameLink signals. These values are stored on seven least significant bits of the control block in the following order: SOF_N – SOP_N – EOP_N – EOF_N – REM (the REM value occupies three least significant bits). Remaining sixty one bits are filled by a sequence of alternating values 0 and 1 starting with

0 at the most sifnigicant bit. The whole padding block is filled by a similar alternating sequence, but starting with 1 at the most significant bit.

Transformation of the FrameLink protocol to the defined types of block to be stored in the dynamic memory is driven by the following set of rules

1. a sequence of FrameLink data inside a FrameLink frame is stored to the dynamic memory as a sequence of data blocks
2. if the combination of values EOP_N='0' & SRC_RDY_N='0' & DST_RDY_N='0' occurs, the control block is stored to the memory
3. the control block is always stored during the same clock cycle as the corresponding data block
4. the total number of blocks representing one FrameLink frame must be divisible by the burst length (i. e. by 4)
5. padding blocks are used for filling gaps between data/control blocks of the same frame or for padding the number of blocks to the burst length boundary

Data transported by the FrameLink protocol can thus be transformed into several different sequences of data/control/padding blocks. All possible sequences are shown in Figure 4.3, where the least significant block is the leftmost one and the most significant bloc is the rightmost one.

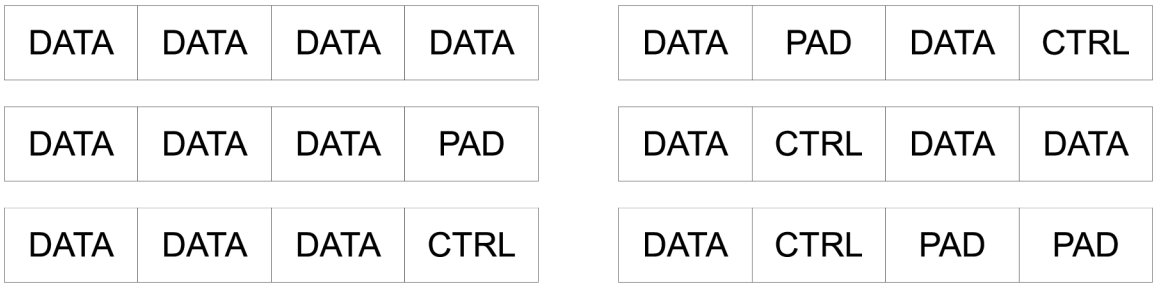



Figure 4.3: Allowed Combinations of the Data Block (DATA), the Control Block (CTRL) and the Padding Block in a Burst

### 4.3.4 Packet Limiter

This part of the application plans and controls data transmission. In this case, the data consist of a Network Layer packet extended by three fields of the Ethernet frame — two of them are the address fields and the last one is the Length/Type field. Based on the selected mode of opreration of the generator, data can come from the Packet Completer module or the Memory Reader/Writer module. An output of the Packet Limiter is always interconnected with the Network Module, as shown in Figure 4.4.

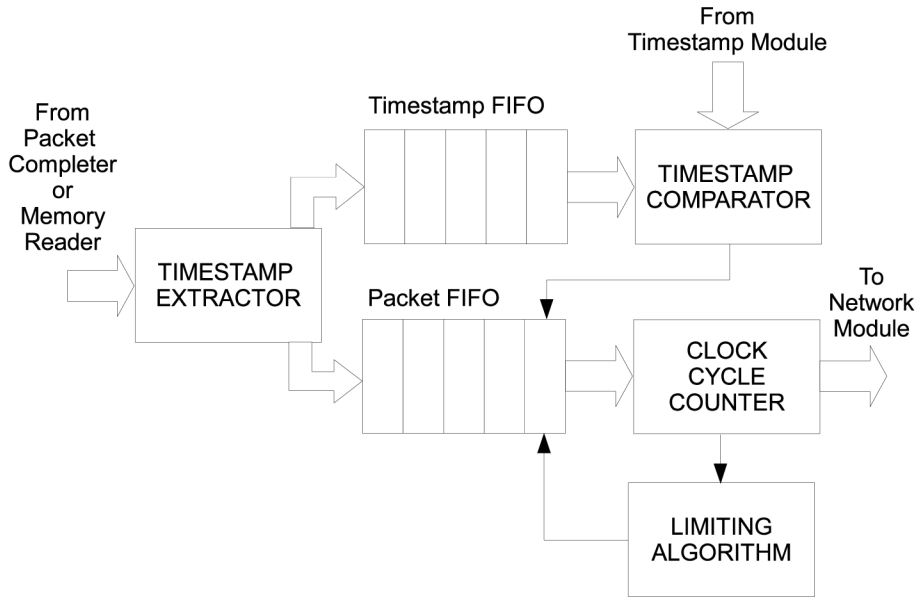



Figure 4.4: Internal structure of the Packet Limiter module

Limitation of data trasmission can be done in three different ways

- no limitation
- limitation to a specified bit rate
- limitation based on 64-bit hardware timestamps

Operation of the Packet Limiter is the simplest when no limitation is applied to data transmission. In such settings, data are simply transmitted to the Network Module as early as they are available. This way of limitation can take place in each mode of operation of the application.

### Limitation to the Specified Bit Rate

Limitation to the specified value of the bit rate can be applied in each mode of operation. It is implemented by the Limiting Algorithm part of the Packet Limiter, which controls reading data from the Packet FIFO. An algorithm of the limitation requires the selected bit rate to be set in a corresponding register. The value in the register specifies the bit rate in Mb/s, but internally it is used as b/$\mu$s. These two values are equivalent and we do not have to perform any conversion.

The limiting algorithm will utilize two counters — the counter of active clock cycles (cycles, when the FrameLink protocol transmit data) and the counter of all clock cycles. These counters are in Figure 4.4 represented by the Clock Cycle Counter part. Each FrameLink frame is trasmitted separately to the Network Module. During its transmission, both counters are incrementing their values. After the transmission of the frame, the transmission of further frames is delayed, but the counter of all clock cycles is still incrementing its value. As soon as the value determined by equation 4.2 gets below the specified bitrate, both counters as well as the algorithm are reseted and the transmission of the next frame

starts. The multiplier 10 000 is a result of converting the number of active clock cycles to the number of transmitted bits in the numerator and the number of all clock cycles to the length of the transmission in microseconds in the denominator. During each active clock cycle, 64 b of data are transmitted and this transmission takes 0.0064 $\mu$s. The fraction $\frac{64}{0.0064}$ gives us the multiplier 10 000 and it also determines the unit b/$\mu$s of the value computed using equation 4.2.

$$\frac{\text{number of active clock cycles}}{\text{number of all clock cycles}} \cdot 10\,000 \tag{4.2}$$

The above described algorithm performs the limitation based on whole frames. First of all, a whole FrameLink frame is transmitted. This frame is followed by a gap ensuring the average bit rate to be the same as the specified bit rate. Chosen approach reflects the architecture of OBUFs, where we can find buffers placed close to the input of the OBUF. This would significantly reduce the effect of the limitation inserting gaps inside FrameLink frames, so the limitation based on whole frames was addopted.

The last note related to the limiting algorithm should be made about size of the counters. Let us consider the worst case, which is specified by a frame of the maximum size (16 kB) and the minimum allowed bit rate (1 Mb/s). Since the speed of transmission is 64 b per clock cycle, we are able to transmit the frame in 2048 cycles. The value of the specified bitrate in Mb/s is the same as in b/$\mu$s. Based on these facts we can compute that for the transmission of the maximum sized frame at the minimum allowed bitrate we will need 20 480 000 clock cycles. This implies the size of the counters to be at least 25 bits.

#### Limitation Based on Timestamps

When transmission limitation based on timestamps takes place, particular FrameLink frames are released from the Packet FIFO at the moment specified by a corresponding timestamp. This way of transmission limitation can be applied only when previously captured real network traffic is replayed back to a network and the traffic is accompanied by timestamps.

Timestamps are usually assigned to frames directly at the time they are received. In order to use the timestamps for transmission limitation, it is necessary to edit them in software (shift them forward in time) before the frames are loaded back to the COMBOv2 card memory. In the Packet Limiter, the timestamps are extracted from the frames and each component is stored in the separate FIFO. The frame is then kept in the FIFO until the corresponding timestamp does not match the currently generated one. If all the timestamps were shifted the same time, this mechanism ensures transmission of the frames with exactly the same time drift as they were captured.

### 4.3.5 Register Control and Status File

A set of registers included in this module serves as a place to store all control and status information of the generator. The most of generator modules use the information stored in registers belonging to this component. This section consists of detailed description of all registers.

In order to clearly present register descriptions, the set of all registers is divided into three groups. The first group consists of general registers, i. e. registers not included in the header fields generating process. The second group contains an overview of registers used

for generating values of fields for the IPv4 header. The last part of the register set contains registers storing values necessary for generating IPv6 header fields.

Each register is considered to utilize 32 b, even when a value stored in the register is not of the size of 32 b. This is often the case and the description of particular registers will focus mainly on clarifying the utilization of bits in the register and specifying meaning of possible values. Not used bits in each register will be considered as reserved for future use and they will be set to 0 by default.

### General Register Set

An overview of this set is presented in Table 4.5 and the following description of each register. Except the name of registers, the table also contains a description of the address space.

| Address | Register |
|---|---|
| 0x00 | Control Register |
| 0x04 | Number of Packets to Be Generated |
| 0x08 | Source MAC Address – Low |
| 0x0C | Source MAC Address – High |
| 0x10 | Destination MAC Address – Low |
| 0x14 | Destination MAC Address – High |
| 0x18 | Payload Pattern |
| 0x1C | Bit Rate |
| 0x20 | Header Fields Generating Mode Register |

Table 4.5: Overview of General Registers and Their Address Space

**Control Register** — there are only two registers storing more than one type of information and the Control Register is one of them. It contains information related to controlling the operation of the aplication. Six least significant bits are used for storing these information and remaining bits of the register are set to 0. Description of used bits and possible values is in Table 4.6.

**Number of Packets to Be Generated** — content of this register specifies in binary the number of frames, which will be generated by the application. The maximum possible number of frames is $2^{32} - 1$ (all bits set to 1). By default, all bits are set to 0, which means unlimited generating of frames.

**Source MAC Address – Low and High** — since a MAC address consists of 48 bits, it has to be stored in two 32-bit registers. The first one will be fully utilized and the second one will store sixteen highest bits of the MAC address. Division of the MAC address between registers Low and High is summarized in Table 4.7. In a case of storing the source MAC address, register names are prefixed by the word *Source*.

**Destination MAC Address – Low and High** — storing a destination MAC address is the same as in the case of the source MAC address (for details see Table 4.7). The only one difference is in the name of registers, which is prefixed by the word *Destination*.

| Name | Position | Values | Meaning | Default |
|---|---|---|---|---|
| Mode of Operation | 0 to 1 | 00 | standard NIC | ✓ |
| | | 01 | loading data to the memory | |
| | | 10 | transmission of data from the memory | |
| | | 11 | generating network traffic | |
| Mode of Limitation | 2 to 3 | 00 | no limitation | ✓ |
| | | 01 | limitation to the specified bit rate | |
| | | 10 | limitation based on 64-bit timestamps | |
| | | 11 | reserved for future use | |
| IPv4/IPv6 | 4 | 0 | generating IPv4 headers | ✓ |
| | | 1 | generating IPv6 headers | |
| DMA/NET | 5 | 0 | loading data from the host | ✓ |
| | | 1 | loading data from the network | |

Table 4.6: Utilization of Bits in the Control Register

| MAC Address Bits | Register Name | Position in Register |
|---|---|---|
| 0 to 31 | MAC Address – Low | 0 to 31 |
| 32 to 47 | MAC Address – High | 0 to 15 |

Table 4.7: Division of the MAC Address Between Two Registers

**Payload Pattern** — this register contains an 8-bit data pattern used by the Packet Completer for filling a packet payload. The pattern is soterd on eight least significant bits of the register and by default, all pattern bits are set to 0.

**Bit Rate** — the value in this register specifies a bit rate to which the Packet Limiter will limit transmitted network traffic (see section 4.3.4). The bit rate is specified in Mb/s and its value can be from the range 1 to 10 000. All other values are not taken into account and in such a case, the bit rate is considered to be set to the maximum allowed value.

**Header Fields Generating Mode Register** — the second register dedicated to store more than one information is the Header Fields Generating Mode Register. Values in this register specifies the way of generating the corresponding IP header field. For each header field, there are two bits whose value define the mode of generating as follows

- 00 — constant value
- 01 — next value from a sequnce of numbers with a given increment
- 10 — random value from a sequence of numbers

Distribution of a register space to different IPv4 and IPv6 header fields is summarized in Table 4.8. By default, all bits of the register are set to 0.

| Name | Position |
|---|---|
| IPv4 Type of Service | 0 to 1 |
| IPv4 Total Length | 2 to 3 |
| IPv4 Flags | 4 to 5 |
| IPv4 Time to Live | 6 to 7 |
| IPv4 Protocol | 8 to 9 |
| IPv4 Source Address | 10 to 11 |
| IPv4 Destination Address | 12 to 13 |
| IPv6 Traffic Class | 14 to 15 |
| IPv6 Flow Label | 16 to 17 |
| IPv6 Payload Length | 18 to 19 |
| IPv6 Hop Limit | 20 to 21 |
| IPv6 Source Address | 22 to 23 |
| IPv6 Destination Address | 24 to 25 |
| Reserved for future use | 26 to 31 |

Table 4.8: Distribution of the Header Fields Generating Mode Register Space Among Different Header Fields

### IPv4 Header Fields Registers

This part presents an overview of registers dedicated for storing values necessary for generating IPv4 header fields. A list of registers together with their addresses is in Table 4.9.

| Base Address | Register Group |
|---|---|
| 0x24 | IPv4 Type of Service Registers |
| 0x30 | IPv4 Total Length Registers |
| 0x3C | IPv4 Flags Registers |
| 0x48 | IPv4 Time to Live Registers |
| 0x54 | IPv4 Protocol Registers |
| 0x60 | IPv4 Source Address Registers |
| 0x6C | IPv4 Destination Address Registers |

Table 4.9: Register Groups Used for Generating IPv4 Header Fields

Since we need to store three important values for each generated IP header field, the set of registers contains three different registers for each field. In order to reduce the size of Table 4.9, each line of the table represents three distinct registers. We can imagine a register schema shown in Table 4.10 to be behind each row of the table.

**IPv4 Type of Service Registers** — as was shown in Table 4.10, this group of registers consists of three registers distinguished by a suffix of their name. Values necessary for generating the Type of Service field occupy six lowest bits of each register. All remaining bits are set to 0 by default.

| Address Offset | Register |
|---|---|
| 0x00 | <*Field Name*> – From |
| 0x04 | <*Field Name*> – To |
| 0x08 | <*Field Name*> – Step |

Table 4.10: Register Schema Which is Behind Each Row of Tables Summarizing Registers for Generating IP Header Fields

**IPv4 Total Length Registers** — each register in this group reserves eleven lowest bits for storing values to be used for generating the Total Length field of the IPv4 header. Allowed values are specified in Table 4.2. Values above the maximum will be considered as the maximum value and values below the minimum will be considered as the minimum value.

**IPv4 Flags Registers** — since there is only one generated bit of the Flags header field, it is sufficient to store only one value, wich will be used in a case of generating constant value of this bit. This value is stored on the lowest bit of the register From. Remaining bits of this register as well as both remaining registers are not utilized and their bits are set to 0.

**IPv4 Time to Live Registers** — values for the Time to Live header field are stored on eight lowest bits of each register. Allowed values are specified in Table 4.2.

**IPv4 Protocol Registers** — a structure of registers in this group is the same as the structure of the previous group of registers. The only one difference is that any possible value on eight lowest bits is considered to be correct.

**IPv4 Source Address Registers** — since an IP address is 32 bits long, all registers in this group are fully utilized.

**IPv4 Destination Address Registers** — this group of registers has the same structure as IPv4 Source Address Registers, but these registers store values for generating the Destination Address field of the IPv4 header.

### IPv6 Header Fields Registers

As well as in Table 4.9, each of the first four rows of Table 4.11 represents three distinct registers, as was shown in Table 4.10. For registers used in generating the Source Address and the Destination Address fields, the schema is explicitly shown. This is because of the need to clarify an order of four registers necessary for representation of each address register. The order is shown in Table 4.12.

**IPv6 Traffic Class Registers** — eight lowest bits in each register of this group are dedicated for storing values for generating the Traffic Class field of the IPv6 header. Each value representable in binary on eight bits is acceptable.

**IPv6 Flow Label Registers** — twenty lowest bits of each register are fully utilized by one of three values important in generating the Flow Label header field. All remaining bits are reserved for future use and set to 0 by default.

| Base Address | Register Group |
|---|---|
| 0x78 | IPv6 Traffic Class Registers |
| 0x84 | IPv6 Flow Label Registers |
| 0x90 | IPv6 Payload Length Registers |
| 0x9C | IPv6 Hop Limit Registers |
| 0xA8 | IPv6 Source Address – From |
| 0xB8 | IPv6 Source Address – To |
| 0xC8 | IPv6 Source Address – Step |
| 0xD8 | IPv6 Destination Address – From |
| 0xE8 | IPv6 Destination Address – To |
| 0xF8 | IPv6 Destination Address – Step |

Table 4.11: Register Groups Used for Generating IPv6 Header Fields

| Address Offset | Range of Bits | Register |
|---|---|---|
| 0x00 | 0 – 31 | <*Register Name*> – 0 |
| 0x04 | 32 – 63 | <*Register Name*> – 1 |
| 0x08 | 63 – 95 | <*Register Name*> – 2 |
| 0x0C | 96 – 127 | <*Register Name*> – 3 |

Table 4.12: Ordering of Four 32-bit Registers Representing One 128-bit Register in Case of IPv6 Header Fields

**IPv6 Payload Length Registers** — allowed values for these registers are specified in Table 4.3. The maximum allowed value can be represented in binary on eleven bits, so the same number of bits is reserved for its storing in each register of this group. Values above the maximum will be considered as the maximum value and values below the minimum will be considered as the minimum value.

**IPv6 Hop Limit Registers** — these registers reserve eight lowest bits for storing values important in generating the Hop Limit field of the IPv6 header. Values allowed on these eight bits are specified in Table 4.2. All remaining bits are set to 0 by default.

**IPv6 Source Address Registers** — since an IPv6 address consists of 128 bits, it has to be stored in four fully utilized registers. The order of registers is specified in Table 4.12.

**IPv6 Destination Address Registers** — the structure of registers dedicated for storing values necessary for generating the Destination Address header field is the same as the structure of IPv6 Source Address Registers group.

### 4.3.6 Main Control FSM

To control the operation of the whole application, the Main Control FSM uses values stored in the Control Register (see section 4.3.5) and dynamically changes application behaviour based on these values. The most important value specifies the mode of operation of the application. Based on this value the Main Control FSM ensures activation and deactivation

of particular generator submodules. The detailed description of all modes of operation is in section 4.2.

Other values stored in the Control Register are used for controlling the specific operation of generator submodules. The Main Control FSM takes these values into account and ensures a proper setting of corresponding submodules.

The value stored in the Mode of Limitation part of the Control Register defines the way of limiting transmitted network traffic. Some modes of limitation are allowed in each mode of operation, some of them can be applied only in specific modes of operation. The Main Control FSM ensures that only allowed modes of network traffic limitation are applied in chosen mode of operation of the application.

The choice whether to generate IPv4 or IPv6 network traffic has an effect on the Pseudo-Random Generator module and the Packet Completer module. The effect takes place only in the generating mode of operation, when the Main Control FSM ensures correct setting of both previously mentioned submodules.

The last part of the Control Register specifies an interconnection inside the Memory Reader/Writer module. This value applies in the loading data to the dynamic memory mode of operation, when it is necessary to determine whether data form the network or from the user will be loaded to the memory. The Main Control FSM ensures the correct interconnection based on the value in the Control Register.

# Chapter 5

# Implementation

The architecture of the application has been thoroughly described in chapter 4. The application was designed with specification requirements (listed at the begining of chapter 4) in mind and in order to provide all specified functions. This chapter describes implementation of the above described design.

Since the application is a firmware for the FPGA chip on the COMBOv2 mother card, the implementation has been done using the *VLSI Hardware Description Language* (VHDL).

Sections within this chapter discuss details of the implementation of distinct submodules and describe decisions made during implementing the generator. At the end of this chapter, there is also a section about tesing of the implementation performed both in a simulation and in real hardware. The chapter is concluded by a section about using the application.

## 5.1 Key Implementation Topics

The aim of this part is to describe in details all important aspects of the implementation, which would not be clear from the text of chapter 4. Sometimes it was necessary to make a decision not discussed during designing the application or to implement some parts different from the proposal. All these key implementation topics should covered and explained in the following sections.

### 5.1.1 MLFSR Pseudo-Random Generators

Discussion about choosing a pseudo-random number generator able to generate a sequence of pseudo-random numbers with good statistical properties and easy to implement in hardware has been done in section 4.3.1. Based on arguments provided in the discussion, a MLFSR has been chosen as the pseudo-random generator for the application.

Implementation of the MLFSR requires specifying of several attributes. While an interconnection of particular LFSRs into the MLFSR is quite clear, attributes of each LFSR have to be chosen very carefully. These attributes include

- a length of each LFSR with respect to a length of the MLFSR
- a primitive polynomial specifying a feedback function (also called taps)
- initial value of a shift register (also called seed)

There are different opinions on the minimal length of LFSRs necessary for building the MLFSR with the maximum period. Although [15] claims, that at least one LFSR must be of at least the same length as the MLFSR, in our application we have implemented each MLFSR of the length $n$ using $n$ LFSRs of the length $n$. With this choice done, it is for sure that the sequence generated by the MLFSR will have the maximum possible period.

If we try to maximize a period of the LFSR, we have to use the suitable feedback function. For example, feedback functions specified by the primitive polynomials (with respect to the length of the LFSR) can be treated as suitable. Coefficients of such primitive polynomials (taps) can be found e. g. in [5]. These taps were addopted also for this application. The feedback function is implemented using the XNOR function and its output is fed to the input of the whole LFSR (Fibonacci scheme).

Since we use the same taps for each LSFR included in the MLFSR, we have to initialize these LFSRs by different seeds. In order to reduce the number of seeds, there is one seed for each MLFSR. This basic seed is used for computation of seeds for particular LFSRs using bit rotation. The basic seed for each MLFSR has been obtained from [11].

### 5.1.2 Generating IPv4 and IPv6 Network Traffic

The MLFSR implemented as described in section 5.1.1 generates the sufficiently random sequence of numbers, but it utilizes a great number of resources growing with the square of a length of a MLFSR output.

The rapidly growing device utilization is not critical when implementing the MLFSR with the output of the length less than 16 bits. Even for the 32-bit long MLFSR (the length of an IPv4 address) the implementation is not a problem. However, problems arise when implementing the 128-bit long MLFSR for generating an IPv6 address. The device utilization by such MLFSR is too high to be implemented together with MLFSRs for other IP header fields.

The above mentioned problem is solved by moving the choice whether to generate IPv4 or IPv6 traffic from run time to design time. This choice is made by setting a generic parameter controlling the application synthesis. The structure of the Control Register described in Table 4.6 stays the same, but the IPv4/IPv6 field is no longer affecting behaviour of the application.

### 5.1.3 Replaying Captured Network Traffic

Implementation of replaying captured network traffic is quite different from the proposed design, so it is important to show what has not been implemented and why.

The design of the application expected utilization of the platform's dynamic memory for storing and subsequent raplaying of network traffic. This memory is connected to the FPGA chip, so it should be possible to access it through the DDR2 Controller module [28]. Correct operation of the controller was shown in a simulation, but accesing the memory through the controller is not working in hardware. Therefore, it is not possible to perform read and write operations on the memory and this part of the application had to be leaved out a this moment.

Since replaying of network traffic from platform's memory could be done without any delays arising from processing network data in the operation system IP stack and transferring them to the moher card over the PCI Express, by not implementing it we loss the most powerful way of replaying previously captured real network traffic. However, it is still possible to use standard software tools for replaying network traffic (`tcpreplay`) when the

application is operating as a standard network interface card. Moreover, the NetCOPE software layer provides tools for fast DMA transfers between the host RAM and the card. This tools can be also used for replaying of previously captured network traffic stored in a PCAP file.

To conclude this section, the implementation of the application allows the user to replay prviously captured real network traffic. It is not possible to accomplish this task by loading data to the platform's dynamic memory and their subsequent raplaying, but it is possible to use standard software tools for packet replaying or the NetCOPE software tools for fast DMA transfers. When using the platform software tools, on some platforms it is possible to replay network traffic at a speed higher than 10 Gb/s (see section 3.3.3).

### 5.1.4 Transmission Limitation

The design proposes three possible modes of limiting data transmission to a network. In the first mode, the Packet Limiter performs no limitation at all. This mode has been implemented. When the last mode of limitation is active, limitation of transferred data is driven by timestamps provided together with transmitted packets. Since this mode of limitation can be active only when transmitted data come from the dynamic memory, it has not been implemented (data to be transmitted cannot come from the dynamic memory — see section 5.1.3).

The last mode to be mentioned provides transmission limitation based on the specified bit rate. In order to accomplish this task, the Packet Limiter should implement the algorithm presented in section 4.3.4. The algorithm is based on two counters of clock cycles and the equation 4.2. In order to map this equation into hardware, division had to be replaced by shifting to the right. The shifting size is determined by a value of the closest integer lower than binary logarithm of the counter value. This approximation underestimates the specified bit rate, so its real value is lower than the bit rate specified by the user.

### 5.1.5 Modes of Operation

Proposed modes of operation of the application can be found in section 4.2. As was already mentioned in section 5.1.3, the implementation does not support storing data to the dynamic memory and their subsequent replay to a network. Not implementing read and write operations on the dynamic memory also implies leaving out two of four proposed modes of operation since functionality designed to be provided in these modes is not available. Because functionality of currently leaved out modes of operation is expected to be implemented in the future, it is still possible to select these modes in the Control Register (see Table 4.6). In such a situation, the application behaves as a standard network interface card. Remaining two modes of operation have been fully implemented.

## 5.2 Application Testing

The main aim of application testing was to identify and remove as many as possible bugs of the implementation.

Testing the implementation in the simulation was performed during the whole implementation process (testing of particular submodules) as well as after it (testing of the whole application). Testing environments including a connection of a tested unit and a description

of a testing scenario are available in respective directories of an implementation directory structure.

Unfortunately, similar test descriptions cannot be provided for testing the implementation in hardware. Since basics of the implementation were tested in the simulation, testing in hardware was focused on proving the correct operation of the application in a target environment. This task was done using software tools described in the next section.

## 5.3 Using the Application

Since any user application based on the NetCOPE platform can be controlled using only software tools provided together with the platform, no software for controlling operation of the applicton has been implemented.

The whole application is driven by values stored in registers of the Register Control and Status File module presented in section 4.3.5. These registers are accessible using the `csbus` tool. The base address of the application's address space is `0x80000`. In the standard NIC mode of operation, network data can be captured and replayed back to a network using the application. These tasks can be accomplished either using standard software tools (`tcpdump` for capturing and `tcpreplay` for replaying) or by NetCOPE platform tools for fast DMA transfers (`sze2read` for capturing network traffic and `sze2write` for its replaying back to the network).

# Chapter 6

# Evaluation

This chapter contains evaluation of the application implemented as described in chapter 5. The chapter comprises of two sections. The first one summarizes typical device utilization by the implemented application. The second section discusses properties of the application from the design point of view as well as from the implementation point of view. These key features of the application are also compared with properties of the packet generator implemented on the NetFPGA platform [8].

## 6.1 Device Utilization

The application has been implemented for the COMBOv2 mother card with attached COMBOI-10G2 interface add-on card. Device utilization by the NetCOPE platform for this configuration of COMBOv2 cards and by the application is shown in Table 6.1. Since the architecture of the application depends on a generic parameter specifying the ability of generating IPv4 or IPv6 network traffic, we have to eximine device utilization for both configurations. Values presented in the table were obtained from results of synthesis for the FPGA chip Virtex-5 LX155T, speed grade -2. The synthesis was done using Xilinx XST, Release 13.1.

| **Resource** | **Registers** | **LUTs** |
|---|---|---|
| Implemented application (one interface, generating IPv4) | 4 276 | 14 875 |
| NetCOPE platform (2x10 Gbit/s interfaces) | 24 383 | 25 420 |
| Summary | 28 659 | 40 295 |
| Available | 97 280 | 97 280 |
| Utilization (%) | 29.46 | 41.42 |
| **Resource** | **Registers** | **LUTs** |
| Implemented application (one interface, generating IPv6) | 35 975 | 50 348 |
| NetCOPE platform (2x10 Gbit/s interfaces) | 24 383 | 25 420 |
| Summary | 60 358 | 75 760 |
| Available | 97 280 | 97 280 |
| Utilization (%) | 62.05 | 77.89 |

Table 6.1: Device Utilization of the Virtex-5 LX155T, Speed Grade -2

From values presented in the table, we can clearly see, that the application, when generating IPv6 traffic, utilizes more than 75 % of available LUTs. These values are therefore another argument supporting the decision to prohibit dynamic selection of generating IPv4 or IPv6 traffic, as disscussed in section 5.1.2.

## 6.2 Application Properties

Although the requirements imposed on this master's thesis were met, the application has been designed to be more general and to be capable of fulfill the set of requirements specified at the begining of chapter 4. Unfortunately, not all of these requirements were met by the implementation, so there is a difference between properties of the design and properties of the implementation. Properties of both the design and the implementation are summarized in Table 6.2.

| **Property** | **Design** | **Implementation** | **NetFPGA** |
|---|---|---|---|
| 10 Gigabit Ethernet support | ✓ | ✓ | ✗ |
| SW based traffic replaying | ✓ | ✓ | ✗ |
| DRAM based traffic replaying | ✓ | ✗ | ✓ |
| Synthetic traffic generation | ✓ | ✓ | ✗ |
| IPv4 support | ✓ | ✓ | ✓ |
| IPv6 support | ✓ | ✓ | ✓ |
| Output rate limitation | ✓ | ✓ | ✓ |
| Timestamp based transmission | ✓ | ✗ | ✗ |

Table 6.2: Comparison of Properties of the Design, the Implementation and the Packet Generator on the NetFPGA Platform

The table also shows properties of the Packet Generator implemented on the NetFPGA platform. Since it is the only existing packet generator based on a hardware acceleration card with the FPGA chip, by comparison of our application with this solution we can show benefits of our solution.

When comparing the NetFPGA packet generator with the implementation of our application, we can see the main contribution in supporting 10GbE standard and in possibility of generating synthetic network traffic. On the other hand, the NetFPGA solution allows replaying of previously captured network traffic using a dynamic memory, but the same functionality with even higher performance can be achieved in our application using replaying of network traffic directly from software.

If we compare the NetFPGA generator with the design of our application, it is clear that the application is designed to be a state of the art packet generator for a hardware acceleration card with the FPGA chip. We must remind, that we are comparing implemented solution with only the design of our application, which is not yet fully implemented. However, this comparison shows a promising future for our application.

# Chapter 7

# Conclusion

The aim of this master's thesis was to design and implement the hardware accelerated network application able to generate synthetic network traffic and to replay previously captured real network traffic. The designed application should be able to perform both tasks at the speed of 10 Gb/s. It was required to implement this application for the COMBOv2 family of cards using the NetCOPE platform.

First of all, a short survey of current existing solutions for generating and replaying network traffic has been done. The survey showed promising possibilities in generating network traffic using hardware acceleration cards with the FPGA chip and thereby confirmed the utility of this thesis. The best parameters of surveyed approaches have been summarized for later use at a design phase of the thesis.

In order to be able to design the application providing specified features, it was necessary to study the OSI Reference Model for network systems, especially the Network and the Data Link layers of this model. It was also important to get acquainted with the family of COMBOv2 cards and the NetCOPE platform, since the COMBOv2 cards are target device family for the proposed application and the NetCOPE platform provides means for rapid development of hardware accelerated network applications.

All the knowledge from previous parts of the thesis have been used for designing the application. The design respects all requirements specified by the thesis itself and it also fulfill some additional requirements, taken mainly from the list of the best parameters of surveyed approaches.

Implementation of the application has been done using VHDL and it meets all basic requirements and also the most of additional ones. The implementation has been tested in a simulation and also in hardware.

It has been shown that a current state of the implementation provides features of at least the same quality as the only existing similar application implemented on the NetFPGA platform. Moreover, some features of our application are even better.

The future work on the application should be focused mainly on allowing the usage of a platform's memory for loading of captured network traffic and its subsequent replaying back to a network. After that, it would also make sense to implement limitation of network data transmission based on timestamps. There are also some parameters of the current solution, which should be improved in the future (device utilization and inaccurate limitation to a specified bit rate). Implementing all proposed improvements and a better software support for using the application would make it a state of the art solution in its category.

# Bibliography


[1] *Information technology — Open Systems Interconnection — Basic Reference Model: The Basic Model*, second edition. ISO/IEC 7498-1:1994(E).

[2] Internet Protocol Suite. Available online [May 2011]: http://en.wikipedia.org/wiki/Internet_Protocol_Suite.

[3] NetFPGA. Available online [May 2011]: http://www.netfpga.org/.

[4] Programmable hardware. Available online [May 2011]: http://www.liberouter.org/.

[5] P. Alfke. *Efficient Shift Registers, LFSR Counters, and Long Pseudo-Random Sequence Generators*. Xilinx, Inc., July 1996. Available online [May 2011]: http://www.xilinx.com/support/documentation/application_notes/xapp052.pdf.

[6] N. Beheshti, Y. Ganjali, M. Ghobadi, N. McKeown, J. Naous, and G. Salmon. Performing Time-Sensitive Network Experiments. In *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS '08, pages 127–128, 2008.

[7] Spirent Communications. Spirent – a leader in test, measurement and service assurance solutions. Available online [May 2011]: http://www.spirent.com/.

[8] G. A. Covington, G. Gibb, J. Lockwood, and N. McKeown. A Packet Generator on the NetFPGA Platform. In *17th IEEE Symposium on Field Programmable Custom Computing Machines, 2009. FCCM '09*, pages 235–238, April 2009.

[9] S. Deering and R. Hinden. *Internet Protocol, Version 6 (IPv6) Specification*, December 1998. RFC 2460.

[10] G. Salmon et al. NetFPGA-based Precise Traffic Generation. In *Proc. of NetFPGA Developers Workshop'09*, 2009.

[11] M. Haahr. Random.org – true random number service. Available online [May 2011]: http://www.random.org/.

[12] The Institute of Electrical and Electronics Engineers, Inc. *Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications*, IEEE std 802.3-2005 edition, December 2005. ISBN 0-7381-4741-9.

[13] Ixia. Ixia – leader in converged IP testing. Available online [May 2011]: http://www.ixiacom.com/.

[14] P. Korček. Pseudorandom Number Generation in FPGA. Bachelor's Thesis, FIT BUT, Brno, 2007.

[15] P. L'Ecuyer and F. Panneton. A New Class of Linear Feedback Shift Register Generators. In *Winter Simulation Conference Proceedings, 2000*, pages 690–696, 2000.

[16] G. Marsaglia. Diehard Battery of Tests of Randomness. Available online [May 2011]: http://www.stat.fsu.edu/pub/diehard/.

[17] T. Martínek and M. Košek. NetCOPE: Platform for Rapid Development of Network Applications. In *11th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems, 2008. DDECS 2008*, pages 1–6, April 2008.

[18] J. Matoušek. Implementation and Verification of Network Interface Blocks. Bachelor's Thesis, FIT BUT, Brno, 2009.

[19] T. Málek, T. Martínek, and J. Kořenek. GICS: Generic Interconnection System. In *International Conference on Field Programmable Logic and Applications, 2008. FPL 2008*, pages 263–268, September 2008.

[20] J. Novotný and M. Žádník. COMBOv2 – Hardware Accelerators for High-Speed Networking, 2008. Available online [May 2011]: http://www.liberouter.org/docs/2008-02-10_COMBOv2.Academic_Forum.pdf.

[21] J. Postel. *Assigned Numbers*. USC - Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, California 90291, September 1981. RFC 790.

[22] J. Postel and J. Reynolds. *Assigned Numbers*. USC - Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, California 90292-6695, October 1994. RFC 1700.

[23] Tcpdump. Web site of tcpdump and libpcap. Available online [May 2011]: http://www.tcpdump.org/.

[24] tcpreplay developers. tcpreplay website. Available online [May 2011]: http://tcpreplay.synfin.net/trac/wiki/tcpreplay.

[25] The Liberouter Project Team. NetCOPE Platform Handbook. Available online [May 2011]: http://www.liberouter.org/netcope/handbook.html.

[26] J. Tobola. Platform for rapid development of network devices. Master's thesis, FIT BUT, Brno, 2007.

[27] USC - Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, California 90291. *Internet Protocol — DARPA Internet Program Protocol Specification*, September 1981. RFC 791.

[28] Xilinx, Inc. *Memory Interface Solutions*, September 2010. Available online [May 2011]: http://www.xilinx.com/support/documentation/ip_documentation/ug086.pdf.

[29] CESNET z. s. p. o. CESNET. Available online [May 2011]: http://www.cesnet.cz/.

[30] CESNET z. s. p. o. Our Hardware — www.liberouter.org. Available online [May 2011]: http://www.liberouter.org/hardware.php?flag=U.

[31] CESNET z. s. p. o. DSC_5459.jpg. Photo from the directory /home/data/foto/Combo-cards/New photos/card 3/.