Czech Technical University in Prague
Faculty of Information Technology
Department of Digital Design

Methodology for Analysis of High-Speed Networks

by

Pavel Benáček

A Doctoral Study Report submitted to
the Faculty of Information Technology,
Czech Technical University in Prague

PhD programme: Informatics

Prague, December 2014

Supervisor:
Doc. Ing. Hana Kubátová,CSc.
Department of Digital Design
Faculty of Information Technology
Czech Technical University in Prague
Thákurova 9
160 00 Prague 6
Czech Republic

Co-Supervisor:
Ing. Viktor Puš, Ph.D.
CESNET, a.l.e.
Zikova 4
160 00 Prague
Czech Republic

Abstract

This report deals with the introduction of the high level synthesis (HLS) in the area of the high-speed computer networks. The proposed methodology is based on the user-defined descriptions of the processing blocks which are connected to the predefined infrastructure of a network device. This approach allows us to control the throughput and resource consumption of the FPGA. Moreover, it enables easy and fast implementation for domain oriented experts. This report also describes three groups of network operations: parsing, classification and data processing. We introduce the state of the art and further discussion for each network operation.

We present already covered area of data processing which allows fast and easy implementation of NetFlow or non-NetFlow actions. This report also presents the implementation of High-Speed Network Stream Merger which is an important entry point to a network device. We also present reached results for these two topics. Finally, the author discusses his future research in proposed area.

Keywords: FPGA, HLS, High-Speed Networks, 100 *Gbps*, Data Parsing, Classification, Data Processing, LPM

Acknowledgment

I would like to thank to doc. Ing Hana Kubátová, CSc. (supervisor, CTU in Prague) and Ing. Viktor Puš, Ph.D. (co-supervisor, CESNET) for their guidance and suggestions during the writing of this report. I also would like to thank to my family, friends and colleagues for their support.

This work is partly supported by the CTU grant "Digital systems design methods for highly reliable and secure systems" No. SGS14/105/OHK3/1T/18, "CESNET Large Infrastructure" project no. LM2010005 funded by the Ministry of Education, Youth and Sports of the Czech Republic and the project TA03010561.

Contents

List of Figures

List of Tables

Abbreviations

CPD	Change-Point Detection
FPGA	Field-Programmable Gate Array
GPPI	Generic Protocol Parser Interface
HDL	Hardware Description Language
HFE	Header Field Extractor
HLS	High Level Synthesis
HW	Hardware
IEEE	Institute of Electrical and Electronics Engineers
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
LPM	Longest Prefix Match
MAC	Media Access Control
MPLS	Multiprotocol Label Switching
NFE	NetFlow Extended
NF	NetFlow
OSI	Open System Interconnection
PE	Processing Element
PP Language	Packet Parsing Language
RR	Round-Robin
SDM	Software Defined Monitoring
SW	Software
TBM	Tree Bitmap
TCAM	Ternary Content-Addressable Memory
TCP	Transport Control Protocol
TD	Timestamp Diff
TFC	TCP Flag Counters
UDP	User Datagram Protocol
VHDL	VHSIC Hardware Description Language
VXLAN	Virtual Extensible Local Area Network
W	Wormhole

Chapter 1

Introduction

Currently wide spread 10 *Gbps* networking technologies are being gradually replaced by 40 and 100 *Gbps* technologies in data centers and on network backbones. In core networks the traffic rate doubles approximately every 18 months [9]. This can eventually lead to the need of technologies faster than 100 *Gbps* very soon. As the speed of traffic convey infrastructure rises, the performance of network processors and networking hardware cannot fall behind and vendors have to create more modern and faster solutions for many network devices like switches, routers or network monitoring probes. The motivation for the technology replacement are services like Video-On-Demand, websites with a video and audio content and growing number of embedded devices with TCP/IP network stack.

The trend of growing network speed is shown in IEEE report [10]. It also predicts that half of backbone network lines will be using speeds 40 and 100 *Gbps* at the end of year 2013. The main result from the IEEE report is shown in Fig. 1.1. The purple bar represents the 40 *Gbps* technology, which will be dominant in data centers. For the administrator of the technology is much better to replace slower technology with faster one. This replacement has impact to power consumption (smaller number of network devices are consuming less energy) and it is also saving cost for cooling of the data center. On the other hand, the administrator of the data center has to count with higher cost of the device.

If we look at Ethernet bandwidths, we find a 10^4 speedup in 27 years - we travelled from 10 *Mbps* in 1983 to 100 *Gbps* in 2010). Moreover, the IEEE group is working on the specification of faster Ethernet interfaces. The new standard will be capable to transfer 400 *Gbps*! You can find more information in the IEEE report [10] and Network Processors study [9] which introduce the evolution of the Network Processors from the beginning to the age of complex high-speed network devices.

The next chapter introduces the background and related work of packet processing and presents actual approaches in the high-speed networking area. Chapter 3 introduces our work in the area which belongs to advanced stateful packet processing and merging of high-speed network streams. Chapter 4 introduces our reached results of introduced approaches and architectures. Finally, the last chapter 5 concludes the work and discuss the future work in next discussed areas.

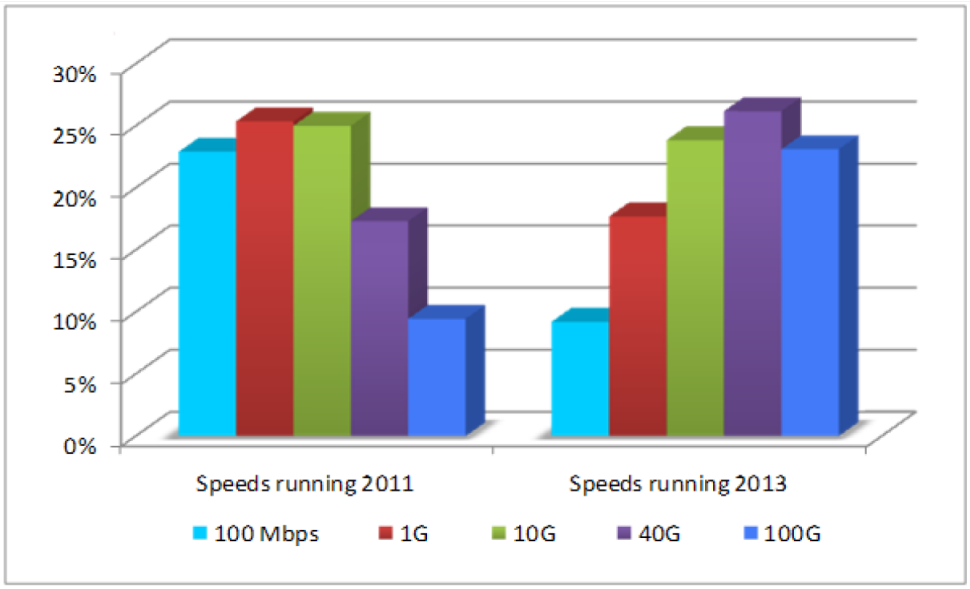


Figure 11—Data center study - percentage of links by speed

Figure 1.1: IEEE 802.3 Industry Connection Ethernet Bandwidth Assessment

Chapter 2

Background and Related Work

Following section discuss the actual approaches in the area of packet parsing, classification and data processing. These operations are very tightly bounded and they need to be solved as effective as possible. Moreover, they are the keys for the effective and fast processing in high-speed network area.

2.1 Basic Issue

In this section, we define three basic network operation groups, which are the core components of each network device. From the packet processing point of view, we can define three basic groups of operations:

1. Data Extraction
2. Classification
3. Data Processing – we will talk about stateful processing

The first group, Data Extraction, is very important for all other groups like classification, routing, filtering of incoming traffic, etc. Each incoming packet must go through some sort of parsing block to understand what is transferred inside the network packet. This examined data is then used in next stages which are related to classification or data processing. Therefore, the parsing is the first action which is performed on incoming network packet and it has to be carefully designed to support expected packet rate of the network interface. For example, the maximal packet rate for 10 $Gbps$ Ethernet with minimal packet length (64 bytes) equals to 1,488,096 packets per second! The network device should be able to process it without problems. It can be otherwise a bottleneck of the system. A simple parsing example can be extraction of the IP address or MAC address.

Classification is tightly bounded to packet parsing and it is typically the second operation in the packet processing chain. The packet classification means categorization of incoming packets into flows. The extracted protocol fields are an identifier, which is used for searching of processing rule. Rule is typically stored in a database of rules. This

database is searched with flow identifier for the best match record. Matched rule contains the identifier of the action, which is performed on the packet. For example, the network switch parse the destination MAC address. This address is used for searching of output network port. This processing block is also very important and it has to be designed on high network speeds. The device should be able to process all incoming packets.

The last stage of the packet processing is the Data Processing block. This block is typically designed to perform some operations like data changing, searching of the Longest Prefix Match which can be used for routing, etc. This stage of packet processing typically accepts read rule from the classification and parsing stage to perform predefined operation which can be set from a software or it can be hard-wired in the networking device. The graphical representation of the packet processing concept is shown in Fig. 2.1.

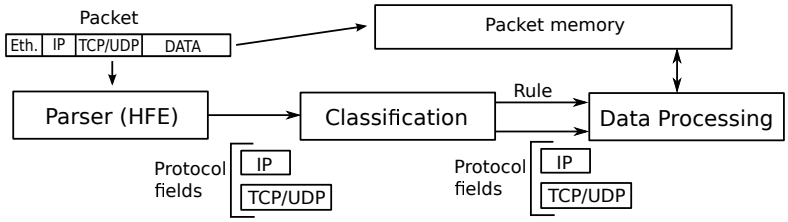


Figure 2.1: Base packet processing pipeline

2.2 Basic Operations

Following text describes the actual State-of-the-Art in briefly introduced groups from the packet processing point of view. It also brings the actual approaches and related work in these areas.

2.2.1 Parsing – Data Extraction Phase

As we mention in previous section, the parsing process is one of the key and important procedures of packet processing. This block of network device is important at each point of network infrastructure. Moreover, there are requests for more complex and non-trivial packet parsing. It is also the key operation for packet classification because results from this stage are used for searching of the rule. The problem is still getting harder because the speed of a networking lines and parsing requirements are still changing very dramatically. The main reason for this dramatic change is growing number of communication protocols. For example, Virtual Extensible LAN (VXLAN) [8] is a network virtualization technology which is used for solving of scalability problems in the area of cloud computing. It is similar to classic VLAN technology [26] but it encapsulates the network packets into UDP packets instead of the MAC-based Ethernet packet (the VLAN ID tag is the part of the Ethernet frame).

Network communication is divided into layers. Each layer has some task which is important for the delivery process of the network packet. The typical structure of the packet consists of a stack of protocol headers. Each protocol adds its own protocol header (we can say metadata) which is understandable by the same network layer on the other device. This fact can be shown in Fig. 2.2.1. The main task of the packet parsing is the extraction of protocols headers, which are added during transition of the conceptual OSI model. Typical extracted headers are IPv4 [21], IPv6 [6] , TCP and UDP headers [22].

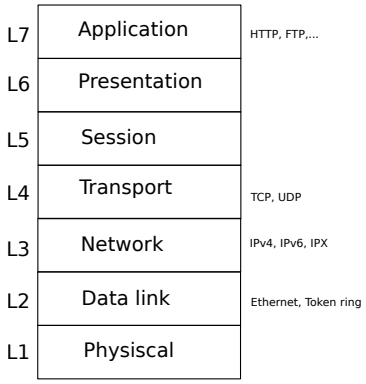


Figure 2.2: Layered OSI Model; It is a conceptual model of network communication. Each layer has some task which is important for success data delivery from one device to other. There is shown the example of main protocols which operates on the layer.

The example of packet processing can be taken from the area of core network. The traffic typically consists of various Ethernet, MPLS, IP and TCP headers. Moreover, there can be a requirement for deep packet inspection of some kind of packets. If we want to run a deep packet inspection, we need to extract and identify the payload and true nature of the packet. From historical point of view, we can define two basic approaches of the parser implementation – software and hardware based parsers.

Software based parsers are the oldest and cheap variant of network parser. The main component is typically the processor which process incoming packets and extracts the interesting data. Parser is typically a computer program which implements this functionality. This approach is also used in these days - mainly for lower network speeds because the final solution is cheap and fast enough for this kind of packet processing. The big advantage of this solution is dynamic behavior which can be accommodate to actual needs (processing program can be easily reprogrammed with new processing program).

The second group, hardware based, is more modern and more powerful solution which is able to process much higher throughput. All modern high-speed network devices contains some special kind of hardware accelerated network parser. Disadvantage of this solution is slower development and higher cost. There is a lot of implementation platforms starting from ASIC to very popular FPGA circuits. The last group of implementation platform is very popular because of its flexibility, programmability and speed. However, hardware based parser is much harder to develop and maintain. This group of network parsers still

contains some challenges to solve. The next section will describe the main approaches of hardware packet parsers implementation.

Braun et al. [4] follows the idea of the layered OSI network model. They introduced a hardware based framework of hand-written wrappers with onion-like structure. For example, the parsing of the IP and UDP protocols consists of parsing of the Ethernet packet, parsing of the IP address and parsing of the UDP protocol. However, the detail description of the interface is not given in the text. The FPGA implementation is able to achieve the throughput of $2.6\,Gb/s$ and it is unclear how the result scales for a wider data paths (e.g. how it scales for higher transfer rates).

Research in this area is very intensive and researches are still focusing on the usage of the High-Level Synthesis (HLS). The HLS is typically the transformation from a higher level description to a lower level description. For example, the Handel-C is very similar to the C language. The HLS description is transformed to the HDL (Hardware Description Language) which is typically used for programming of the FPGA circuits. There is a general result for the HLS approach which was given by Dedek et al. in [5]. He investigated three different realizations of the same block from different point of view. There is comparison of two different realizations in embedded processors (software implementation in customized 16-bit RISC and soft-core $MicroBlaze^{TM}$ processor) and specialized hardware based implementation with usage of Handel-C language. The comparison was made from different point of views: the development time, the estimated frequency and the occupied area. This work demonstrates that using of processors gives poor results. The average throughput equals to $712\,Mb/s$ in the case of custom RISC processor, $83\,Mb/s$ for the $MicroBlaze^{TM}$ and $1454\,Mb/s$ for Handel-C implementation. The second important demonstration is coming from the usage of the HLS – the throughput and short development time can be obtained by usage of the HLS (Handel-C in Dedek's example).

Kobiersky et al. [14] introduces the packet header analysis and field extraction for multigigabit networks. The architecture for packet header processing is generated from standard XML protocol scheme. More than $20\,Gbps$ throughput can be achieved using less than 5000 slices of Virtex 5 FPGA. Maximal frequency depends on internal crossbar which doesn't scale well for wider data widths. However, it is important result which shows the approach of High-Level Synthesized network parsers for High-Speed networks. Moreover, it allows us to dynamically react on new network protocols. Unfortunately, this approach is harder to use in faster network solutions (like 40 and $100\,Gbps$).

A good example of usage of HLS was given by Attig and Brebner [2]. They introduce a PP language – a simple high-level language for description of packet parsing algorithms in an implementation-independent manner. The approach presents the implementation for modern FPGA which is able to accommodate modern high-speed network processing. Compilation of the PP involves of virtual processing architecture which is tailored to a specific packer parsing requirements. The PP language describes the structure of the packet headers and methods which define parsing rules. The system scales from 1 to $400\,Gbps$ network speeds on Virtex-7 FPGA. However, the basic architectural structure of the Brebner's parser is massive pipeline which consumes about 10% of Virtex-7 resources for 1024 bits wide datapath. However, the pipeline increase the latency of the parsing

process. This is inconvenient in some applications like stock exchange algorithms where the latency is crucial.

Puš et al. [25] proposed a novel architecture of a pipelined packet parser for FPGA based solutions, which offers low latency in addition to high throughput. This two parameters can be tuned to the needs of a particular application. Usage of such solution is very wide. The parser is hand-optimized thanks to direct implementation in VHDL. The parser consists of analyzing blocks for variety of protocols like IPv4, IPv6, MPLS, etc. Each block perform analysis of one protocol – it allows the main idea of the OSI protocol which was discussed in previous text. Moreover, the communication interface of parsing blocks is generally defined and it can be used for new protocols. *Generic Protocol Parser Interface*(GPII) consists of the input information which is needed for parsing of single protocol. GPPI output interface includes the parsed data and information for the next stage parser. However, the actual implementation doesn't contain any HLS support. It can be interesting to define a description of the protocol which will be usable for transformation from the general description of the parsing process to the described hardware architecture. The parser is able to consume only 1.19% of Virtex-7 FPGA resources to achieve throughput over 100 *Gbps* and 4.88% to achieve throughput over 400 *Gbps*.

2.2.2 Classification Phase

Classification is tightly bounded to packet parsing and it is typically the second operation in the packet processing chain. The packet classification means categorization of incoming packets into flows. The flow is an identifier, which is used for searching of processing rule. Rule is typically stored in a database of rules. This database is searched with flow identicator for the best match record. Matched rule contains the identifier of the action, which is performed on the packet. The performance of the packet classification is important parameter of the final throughput and all algorithms and architectures have to be designed for this task.

Balasaheb and Kulkuhari [1] have proposed four main groups of packet classification approaches: exhaustive search, decomposition, decision tree and hierarchical-tire (H-trie). The packet classification can be viewed as a geometrical problem of searching in multi-dimensional discrete space. Each dimension can be understand as one parameter which is used for the classification. For example, the most common classification is using five dimensions – source and destination IP addresses, source and destination ports and type of transport protocol. If we combine all these parameters together, we will have a five-dimensional search space where each point defines one packet (e.g. one combination). We can assign a rule to each point (or points). The process of packet classification from all rules containing (matching) the packet selects the one with the highest priority.

Any searching problem can be solved by searching through all available entries. We can identify two basic approaches – linear search and TCAM based search. The first group ,linear approach, is performed by comparing of the parsed header with all the entries in the list (memory). The disadvantage of this approach becomes slow for large set of rules because the asymptotic complexity of the linear probing is $\mathcal{O}(n)$. We can use a TCAM for

faster searching because this kind of memory is able to search through all rules in parallel. However, the complexity, cost and power consumption is growing with a number of inserted rules. More details about this approach can be found in [24].

The decomposition based solutions are using the Cartesian product on independent searches on each header field. These approaches offer high throughput but require a big amount of storage space. The primary challenge is how to efficiently aggregate the results of the single field searches. The required storage space can be reduced by usage of pseudorules which minimize the result of the Cartesian products. The pseudorule is a generated rule which handles situations for packets which are not directly defined by the rule but it is needed for success process of the classification. However, this set of rules and pseudorules can be quite large. For the minimization of generated pseudorules, the Prefix Filtering Classification and Prefix Coloring Classification algorithm was introduced in [24]. These approaches are able to minimize and eliminate a big amount of pseudorules. The classification hardware can be faster and it can accommodate bigger amount of rules and needed pseudorules.

Most of the classification architectures and approaches are based on direct mapping of this problematic to decision trees, which represents the geometrical approach of this problem. HiCuts and HyperCuts are the main representatives of this approach. They construct a decision tree where inner tree nodes divide the space by hyperplanes into subspaces (e.g. it "cuts" the space into small subspaces which will be searched for the rule). The HyperCuts algorithm is faster because it allows cutting on multiple fields per one step. However, the depth of the tree depends on a particular set because the worst case represents the longest path from a root to a leaf. The throughput strongly depends on the passed rule set (e.g. generated tree).

The H-Trie is using two group of prefixes – source IP address and destination IP address prefixes. The initial trie is constructed by usage of the source IP address prefixes. For each source IP address prefix node is constructed the destination IP address prefix trie with associated source IP address prefix. Therefore, the H-Trie consists of initial source IP address prefix trie with connected destination IP address tries. Search starts from the source IP address trie and continues with destination IP address trie until the desired rule is reached in a leaf. This is the main idea of Grid-of-Tries [24].

2.2.3 Data Processing Phase

The data processing phase is typically the last stage of packet processing which is made after parsing and classification. The data processing can be a general operation which is related to data change, update of the network flow, etc. In this work, we will talk about update operations which are related to traffic measurement because this stage of the packet processing is strongly application specific and it can differ from application to application. Moreover, this application is very important for network security, real-time traffic analysis or services like accounting.

Fundamentally, the traffic measurement involves counting number of packets (e.g.update operation) which meet some criteria. Moreover, the update criteria can be much more

complex. For example, we can use statistical methods for detection of anomalous traffic. The traffic is measured in terms of *flows*. The flow refers to a set of packets which have the same n-tuple in their header field. The n-tuple typically consists of five members: $\{sip, dip, spt, dpt, prt\}$ where, *sip* and *dip* stands for source and destination IP, *spt* and *dpt* stands for source and destination port. Finally, the *prt* stands for the type of L4 transport protocol – TCP or UDP.

Following text describes technologies for monitoring of computer networks which is needed in many networking fields like accounting, computer security or monitoring of the network infrastructure. From the data processing point of view, we can identify two basic approaches – *Per-flow statistics collection* and *Query-driven measurement*. Following text describes the basic overview of mentioned approaches.

2.2.3.1 Query-driven measurement

This approach was introduced by Khan et al. in [13]. The key issue of the Per-flow statistics collection is lack of better scalability for real-time traffic measurement. If we want to pass through these issues, we have to leave the classic approach and use the *Query-driven measurement*. Simply said, we can understand it as a real-time measuring machine where the query is directly programmed into the monitoring devices. Passing data are searched for designated query which is good for latency and it is suitable for high-speed network data. The disadvantage of this approach is complexity of final hardware if we choose complex update operations. For this case, new hardware architectures and algorithms has to be introduced. Processing logic of the proposed architecture consists of many simple processing elements (PE) which are connected into the chain. Each element can be programmed to perform some operation. Finally, the solution were able to process traffic at speed of 10 *Gbps*. The solution scales sub-linear with the increase in number and size of the PE. Unfortunately, the authors of the paper didn't talk about scalability for faster interfaces. The brief example of proposed architecture is shown in Fig. 2.2.3.1

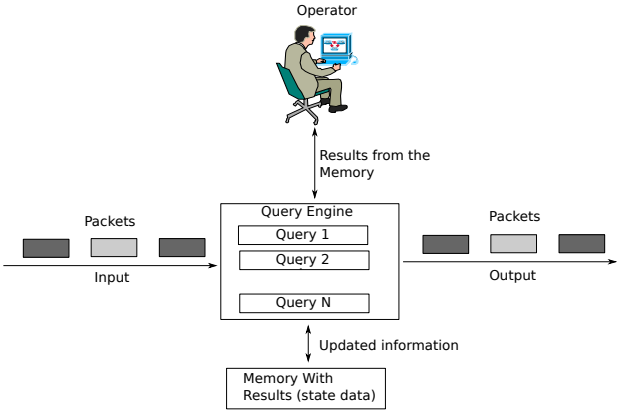


Figure 2.3: Brief overview of the Query-driven measurement

The Gorilla project [15] is the nearest comparable SDM-like solution. Gorilla is a

methodology for generating FPGA-based solutions especially well suited for data parallel applications. The main goal of Gorilla is the same as our goal in SDM Update – to make the hardware design process easier and faster. Our solution is however specially designed for the stateful processing of network packet data Furthermore, SDM is able to work with L2–L7 layers of ISO/OSI model.

2.2.3.2 Per-flow statistics collection

This approach can be referred as a traditional measurement scheme. It is based on storage of per-flow based on high-density media (storage for example). This stored data is processed later to find the answer on the query. The most known and used technology was developed by CISCO and it is known as a NetFlow.

This technology requires three components – NetFlow Exporter, NetFlow collector and NetFlow statistics storage. NetFlow statistics are sampled from network devices such routers, switches or network probes, to network collector.

We can identify two basic monitoring approaches – active and passive monitoring. The active monitoring is related to measuring on active network devices like routers or switches. It is the additional functionality which consumes available resources. For example, the main functionality of a router is routing of network packets and monitoring consumes the processing time which is crucial parameter in the high-speed networking area. This can be problem during a Denial-of-Service attack where is the number of incoming packets rapidly increasing.

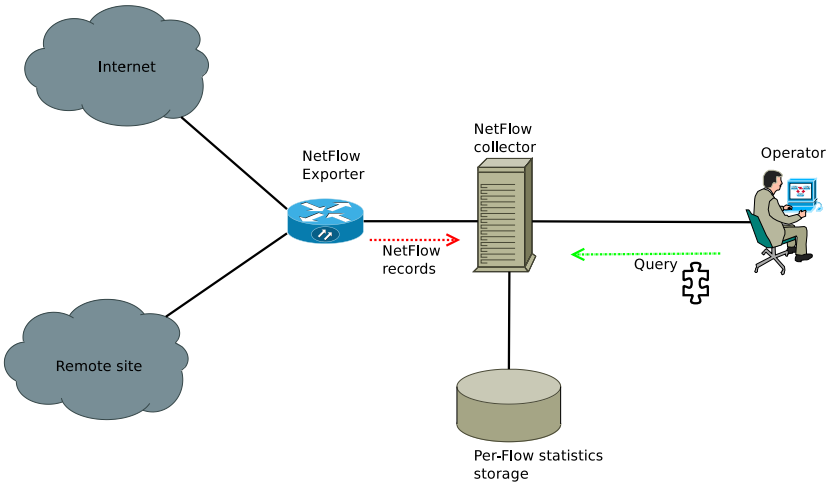


Figure 2.4: Brief NetFlow architecture

The passive monitoring is based on a stand-alone network devices which is known as a network probe. The probe is specialized devices which is designed for monitoring purposes and it can be tailored to the networking needs. The disadvantage of this solution is the need of a new networking box in the data center. However, the purpose of this approach is as same as the active monitoring – aggregation of network statistics.

All aggregated statistics are stored by the NetFlow Collector to the flow storage. Typical timeout for statistics collection equals to five minutes. This timeout provides satisfiable resolution and memory requirements. The most modern collectors and exporters support insertion of information from application layer (L7) which makes user queries much more complex. Discussed architecture is shown in Fig. 2.4. Measured statistics are processed by usage of queries which are created by the operator. The answer of the query is computed on a NetFlow Collector (for example a number of received bytes in flow with specific IP address and source TCP port). The main disadvantage can be a higher latency of detection.

The latency can be tuned by the timeout but low value of the timeout can be counterproductive because there will be much more data to process. It is important to find the good compromise between latency and detail of the measurement. Moreover, this approach requires computation intensive processing and I/0 overhead. As a result, the scalability of this approach for high-speed networks is not good enough. As speed and complexity of the computer network rises, the new approach has to be introduced and it is described in next section of this text.

The *Software Defined Monitoring* was introduced by Kekely, Puš and Kořenek in [12]. It is the example of advanced flow based monitoring with hardware accelerated extension. The main idea of this approach is to offload the uninteresting flows into the hardware accelerated analyzer while the interesting flows are sent to the software for deeper and more complex analysis. It eliminates the lack of scalability for faster interfaces. Moreover, it connects the world of high-speed networking and high-level synthesis. This approach was demonstrated by Puš and Benáček in [23]. The paper demonstrates the ability of FPGAs and high-level synthesis (HLS) tools available today to extend the functionality of the hardware by new instructions – new types of aggregation functions. While the typical use of HLS is the synthesis of digital signal processing algorithms, they employ a C/C++ to VHDL synthesizer to implement the instructions of the application specific processor. Authors of the paper implemented the processor infrastructure in VHDL and prepare it for a wide variety of instructions. Finally, they evaluate the approach by implementing five different instructions (two of them being NetFlow like, three non-NetFlow) in high level language (HLL) to assess the versatility of the HLS for the task of processor instruction description. The SDM consists of two main parts (firmware and software) connected together via PCI Express bus. Both parts are tightly coupled together to allow a precise control of the hardware processing on the per-flow basis. The software part of SDM consists of the controller and monitoring applications. The advanced monitoring tasks, such as analysis of application protocols is performed in the monitoring applications. The management of the hardware (removing and insertion of the processing rules) is performed in the controller. The SDM Processor hardware is controlled by instructions stored in the rules. The instruction tells the hardware what action must be performed for each input packet. The hardware passes the data to the software in the form of packet metadata (extracted information from the packet header) or aggregated flow records (NetFlow for example). Whole received packet can be also sent to software for deeper analysis. The hardware part is described in detail in Sec. 3.2.1. This section belongs to our approach of

high-level synthesis in the high-speed networking world – we called it as SDM Update.

The SDM hardware is controlled by instructions stored in the rules. The instructions tell the hardware what action must be performed for each input packet. After executing the instruction, the hardware passes the output data (if any) to the software in the form of packet metadata (extracted information from the packet header) or aggregated flow records (NetFlow for example). Whole received packet can be also sent to software for deeper analysis. Graphical representation of the SDM concept is shown in Figure 2.5.

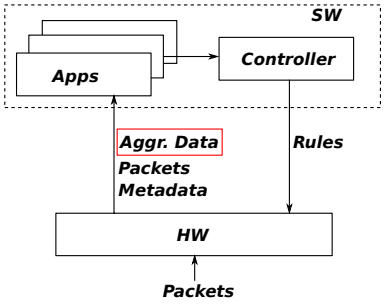


Figure 2.5: Brief SDM Architecture

2.2.3.3 Generalization of the Update Operation

Previous sections was talking about the NetFlow statistics update. This operation is well defined and it consists of following steps:

1. Update of the start and stop timestamp
2. Update of the number of transferred bytes
3. Perform the logical OR operation on incoming TCP flags (e.g. it remebers all used TCP flags over the life of the flow)

The NetFlow operation is statically defined and used in many networking devices. It can be used for many basic flow analysis. However, this basic analysis is not good enough for real-time base measuring like CPD (Change-Point Detectoin) or complex analysis high-speed network analysis in the hardware.

We can generalize the update process and create the user defined operation for each flow. We are mainly talking about stateful data processing – we remember some state from previous run. In general, we need to define the update process and structure of the record. This approach was shown in the SDM Proccesor 3.2.1 which is following this process generalization idea. The advantage of this solution is the flexibility and possibility to implement modern monitoring approaches. The next advantage is the usage of the HLS language which allows us to use knowledges of domain oriented experts.

Chapter 3

Overview of Our Approach

In previous chapter, we were talking about actual approaches and in several areas of high-speed network processing. We introduced all three important operations – parsing, classification and advanced data processing. Following section introduces overview of our approach in advanced data processing area.

3.1 Basic Approach – Data Processing Phase

In this section, we will introduce our basic approach in this area of high-speed network processing. As we mention, the main problem of high-level synthesized (HLS) solutions is the throughput of the final solution. The advantage of the HLS approach is the simplicity of the development. Moreover, the time needed for the development is typically decreasing. The second, very common, approach is based on the handwritten designs. This approach brings better control of final design. Finally, the programmer can tailor the design with less resource consumption.

In our approach, we intend to connect these two worlds together. Our approach is able to connect the HLS world with handwritten one. This allows us to tune the final throughput and used area of the FPGA chip. This approach consists of:

1. Definition of communication interfaces
2. Definition of the general infrastructure which will support user defined operations
3. Definition of handwritten blocks (e.g. basic component library)
4. Definition of parameters for HLS defined user operations (e.g. frequency and Initiation Interval)

The first group of definitions is important for interconnection of components of the data processing modules and it has to be carefully selected because it will be used by all remaining definitions. The second and third group of definitions are very tightly connected and it is the core of our approach. All handwritten modules are carefully designed for high

throughput and low latency. They are tailored to the high-speed world with regard to consumed resources of the FPGA. The last group of definitions is important for the final throughput of the data processing module. We can identify and define two basic parameters of the HLS defined user operation – Initiation Interval (II) and frequency. The II parameter represents the time needed for initialization of a new request in the processing system. The frequency stands for a number of cycles during one second.

We will demonstrate one architecture in the following text where will be described the architecture of such Data Processing unit. We chose the SDM [12] as our implementation platform. The following text also describes the advanced architecture for merging of high-speed network streams. This component can be used for aggregation of more network streams into one stream which can be processed in described SDM Data Processing unit.

3.2 Architecture of the Data Processing Unit

The basic approach and top-level architecture was described in Sec. 2.2.3.1. The following text is focused on detail description of SDM Processor which is the key element for the stateful packet processing.

3.2.1 SDM Processor Description

For the scope of this work, the hardware part of SDM is the most interesting. SDM Processor is conceptually an application specific network processor for a stateful flow-based network traffic measurement. The main parts of the SDM Processor are shown in Figure 3.1. The processing of all incoming packets starts with header parsing and extraction of interesting metadata in the Header Field Extractor (HFE) block. This prepared data is passed to the Search module which performs lookup in the Rule Memory (these rules are maintained by the software SDM controller). If the rule is found, it is sent with the data to the SDM Update module which performs the operation with respect to the passed instruction. If the rule is not found, default behavior is performed - packet is typically passed to the software for further analysis.

3.2.2 Structure of SDM Instruction

The instructions are stored in the Rule Memory, together with the flow identification. The Search module finds the appropriate instruction for each packet and passes is to the Update module.

The instruction structure is shown in Figure 3.2 and it consists of four parts, with the total length of 72 bits. The opcode fields are used for the identification of the update operation. OPCODE1 field is used for the request routing in the Instruction Decoder. OPCODE2 can be used in the instruction block for further specification of update/export operation. This field is not used during the routing process.

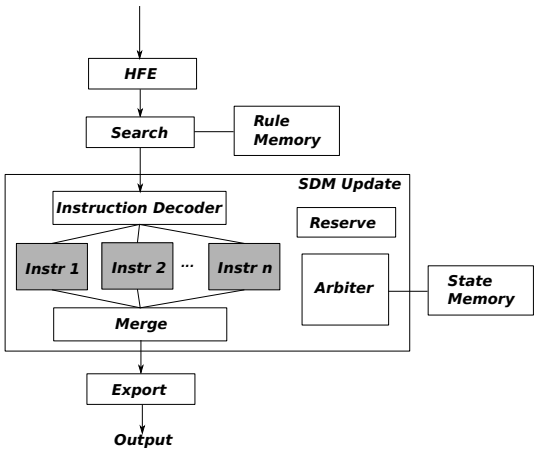


Figure 3.1: SDM firmware architecture and details of SDM Update module

The PARAMS field is important for the SDM Update processor because this field holds parameters like the address of the record in memory. This field can be used for passing the additional parameters from the software, because the monitoring application writes to the Rule Memory via the SDM Controller (see Fig. 2.5).

The ACTION field is not directly used by the instruction blocks, it is rather used by components connected after the SDM Update module (the Export module, see Fig. 3.1). This field is used for the specification of the operation after the update. Typical actions are: export the whole packet to software, export the medatada to software, drop the packet, forward the packet back to the network.

OPCODE1 (4 bits)	OPCODE2 (4 bits)	PARAMS (32 bits)	ACTION (32 bits)

Figure 3.2: Instruction structure

3.2.3 SDM Update

The SDM Update module brings the statefulness to the SDM Processor. It manages the flow records – updates the aggregated data in the State Memory. The module mainly updates the stored values based on the input data and the rule instruction. The instruction for every packet has the address of the record and a specification of the operation (aggregation type). Update of the record is delimited by two memory operations: read the stored values of the record and write back the updated values. Special type of operation is the export of the record values, possibly followed by clearing the record values in the memory. Records can be exported not only at the flow end, but also in periodical manner, so that the software applications can have actual information about hardware monitored flows. Control of memory allocation for records and their periodical export is realized by the SDM controller software.

Infrastructure around the instruction blocks is designed with respect to easy implementation of new instructions. Blocks of this generalized environment (created specially for network purposes) are shown in Figure 3.1. The main services of this infrastructure are:

- Request decoding and forwarding to the appropriate instruction processing module. Forwarding decision is based on the instruction opcode and a simple routing table that is stored in the Instruction Decoder block.
- Memory record reservation (to avoid RAW data hazards)
- Memory access arbitration
- Output synchronization (to preserve the ordering of requests).

3.2.4 Description of Instruction Block

Detailed schematic of *Instr #* block is shown in Figure 3.3. Each instruction block consist of a few basic building blocks, which are written by hand and are also part of the SDM VHDL library. The processing of an input instruction request is forwarded by the Instruction Decoder to the appropriate *Instr #* block. The instruction enters the module which takes care of the reading and reservation of the record stored in State Memory. The address of the record is part of the input instruction. Read data is passed together with the instruction to the Update pipeline where the HLL synthesized block is located. This user-defined block is surrounded by the SDM interface adapter which is used for conversion from internal SDM Update interface to predefined user instruction interface.

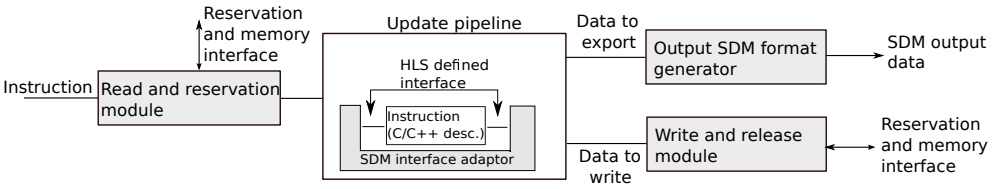


Figure 3.3: Details of instruction implementation

3.3 Architecture of High-Speed Network Merger

The main motivation for this architecture is shown in Fig. 3.4. This picture shows the brief design of a Network Interface Card, which is equipped with two 40 *Gbps* ports and one processing engine. Processed data are then transfered via DMA channels to the RAM where can be processed by CPUs. The main problem is the merging process of incoming network data from two 40 *Gbps* to one 80 *Gbps* line!

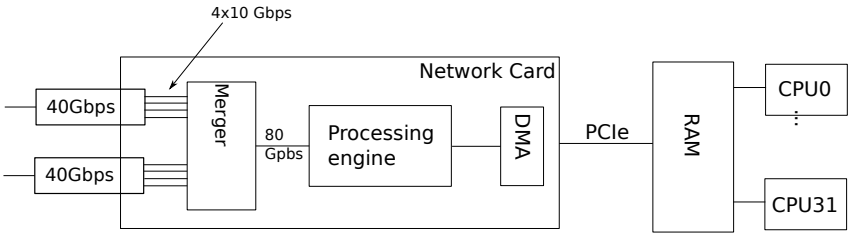


Figure 3.4: Motivation for High-Speed Network Merger

3.3.1 Theoretical Requirements

The main hardware architecture requirement is the throughput which has to be higher or equal to the sum of speeds of all available ports. In our case, we need to transfer at least $80\,Gbps$. Firstly, we need to specify the parameters of the binder's output bus. There are three main parameters — bus width, frequency and efficiency of the data transfer.

The *Bus width* means the number of bits that can be transferred during one bus cycle, it is typically the power of 2 and it also describes how many signals are used for parallel data transfer. The *Frequency* tells us the number of bus cycles that can be performed during one second. Roughly said, theoretical upper bound of throughput can be computed as:

$$throughput = buswidth * frequency\,[bps] \tag{3.1}$$

Unfortunately, this statement is true if and only if we use the whole bus width for the data transfer (e.g. no free space is available on the bus during a data transfer). This isn't true in the Ethernet world because the standard defines the Ethernet frame length which is in range from 64 to 1518 bytes. Moreover, the standard also supports Jumbo Frames which transfer more than 1500 bytes of payload. Therefore, the bus need not to be effectively utilized every clock cycle.

The third parameter is the *efficiency* of a data transfer. We can define bus efficiency as the percentual usage of the bus during transfer of the end of actual frame and start of a new transferred frame. There are two possible implementations of the start of the frame signalization:

- **Aligned** - Next transferred frame starts at the fixed position which typically equals to the first byte of a next bus cycle. The advantage of this solution is simplicity and powerful hardware design because operations like data shifting, block reordering can be performed in the next bus cycle (they can be easily pipelined). The disadvantage of this approach is not effective data transfer during the end of the frame which occupies only one byte of the available bus capacity. In this case, we are using only the fraction of all available resources and the throughput is decreasing.

- **Unaligned** - Next transferred frame starts in the same bus cycle where the end of the previous frame is active (e.g. the bus word is shared by two frames). The start of

the frame can be signalized on several positions in the shared bus word (depending on the granularity of signalization). The advantage of this solution is an average efficiency during the data transfer because unused place for data is smaller than in case of aligned version.

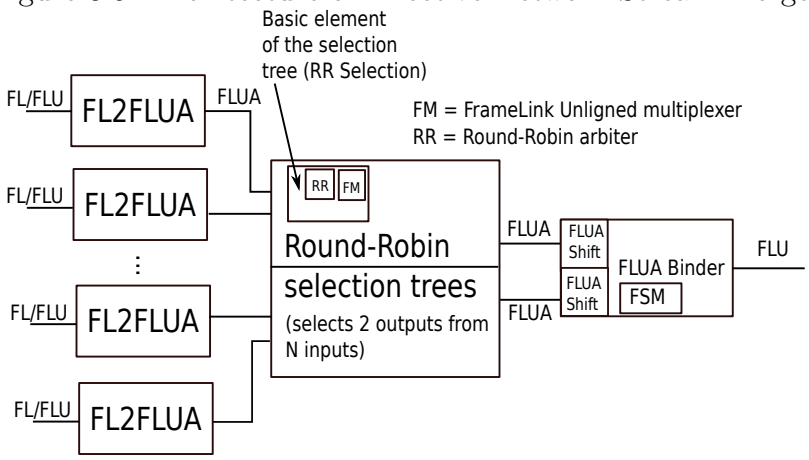
Roughly said, the maximal throughput of the bus will be 102.4 *Gbps*. The unaligned version of the protocol was selected for better efficiency of the bus during a data transfer (we want to utilize the bus as much as possible). As we mention in the text before, the unaligned version of communication protocol is able to transfer the end of previous frame and the start of the next frame in the same bus cycle. Therefore, if we want to merge all the input high-speed network streams, we need to select two frames and merge them into shared output bus. We divide input ports into two groups where two independent frames will be selected and effectively merged into output bus. Therefore, each group should contribute to the output throughput by half.

3.3.2 Detail Description

The proposed architecture of effective network stream merger is shown in Fig. 3.5. Following the main idea from previous section, we divided all input ports into two groups. Each group will contribute to the throughput by half. For output merging, we intend to use the aligned version of the protocol because we know where the processed frame starts – we call this version of protocol as FLUA (FLU Aligned). Therefore, we need to prepare the start of the new processed frame behind the end of the previous transferred frame. Due to these facts, the port group should minimally generate the half of desired throughput. Notice, the aligned version of communication protocol is used from the output of FL2FLUA block to FLUA Shift unit. The shortest frame on Ethernet, where the transfer is ineffective, equals to 65 bytes. Transfer of this packet takes two bus cycles where the last bust cycle consists of one byte. Roughly said, throughput of such port group can be 50 *Gbps* (one packet per two bus cycles). This is enough to accommodate the half of desired throughput. If we want to add another network stream, we should utilize the second port group which is connected to the second FLUA Shifter module which is able to shift and arrange the network stream to our needs. Therefore, the effective merging can be performed to create the unaligned high-speed version of output protocol (denoted as FLU after FLUA Binder in Fig. 3.5).

All input packets have to be transformed to the aligned version of the communication protocol which is called the FLUA (e.g. aligned version of the FLU protocol). This modified stream is important to the FLUA Shifter because this module performs shifting operation with predefined offset. This offset has to be set at the beginning of a new transferred frame. This shift operation helps us to merge two frames (from distinct port groups) into unaligned protocol version which is more effective because the next transferred frame starts in the actual bus cycle. In the case of unaligned version with shared word, we have to split the effective flow into two independent flows which will be aligned (e.g. we have to deal with transfer of two frames during one bus cycle). This operation helps us to transfer

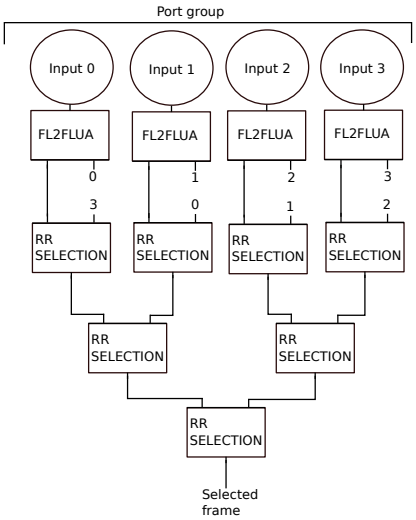
Figure 3.5: Architecture of Effective Network Stream Merger



the FLU protocol into FLUA protocol which is used by the FLUA Binder module (see Fig. 3.5).

Modified and translated flows are then transferred to the FLUA Binder via Round-Robin selection trees (see Fig 3.5). This module consists of two selection trees because we are operating with two port groups. The detail structure of one selection tree is shown in Fig. 3.6. The RR SELECTION block is a multiplexer with Round-Robin arbitration of input ports.

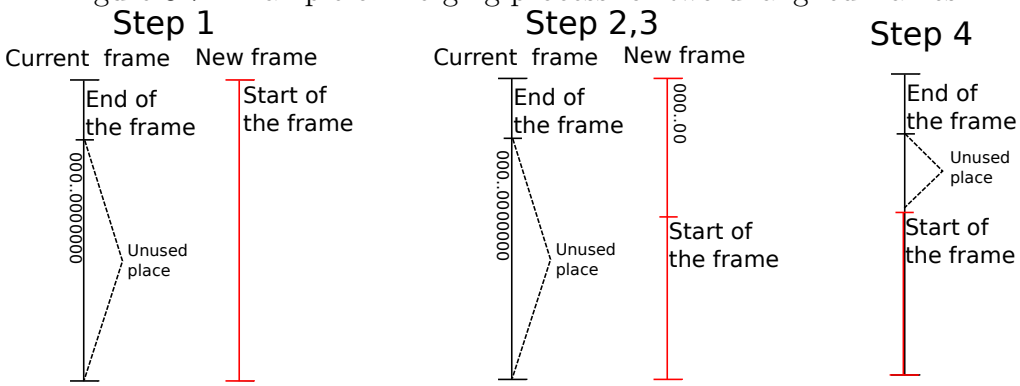
Figure 3.6: Architecture of Round-Robin selection tree; Some lines are omitted for brevity – outputs and inputs are matched by numbers



The tree height is equal to the number of RR SELECTION nodes on the path from the root (output) to a leaf(input port). The height equals to $\log_2(2 * PORTS)$ where $PORTS$ is the number of ports in a port group. Each RR SELECTION module can be equipped with register which is used for better timing result. This architecture of selection modules

is scalable because doubled number of input ports infer one more RR SELECTION layer which can be equipped with registers for better timing results. Due to these facts, we can accommodate more input ports and achieve very similar frequency of the final solution.

Figure 3.7: Example of merging process for two unaligned frames



The last module is the FLUA Binder which transforms two FLUA flows into one more effective FLU protocol. This module consists of two FLUA Shifter modules and the control Finite-State Machine (FSM). The behavior of FLUA Shifter was mentioned earlier in the text. The most important module (in the FLUA Binder) is the FSM because this block controls merging process and everything around. It also contains a table of configurations for FLUA Shifter modules. The table is indexed by the position of the end of actually transferred frame. This table of shift values can be generated from the passed configuration of output bus during the translation time. This approach helps us to reach better timing results because we can store the table in the form of Read-Only Memory (ROM). Due to these facts, the FLUA Binder module is ready to accept new output bus configuration to prepare its structure for a new environment. The FLUA Binder module complexity stays unchanged with growing number of input ports – complexity of this module depends on configuration of output bus. Finally, the example of the merging process of two frames is shown in Fig. 3.7. For successful merging process, we need to modify input frames and met these requirements:

1. Fill the unused place of currently transfered frame with log. 0
2. Fill the unused place of the new frame with log. 0
3. Fill unused control signals with log. 0

After that, the merging operation can be performed with logical OR operation which is easy to perform on wide data busses and it can be easily pipelined for higher performance of the final design. Finally, we can define the basic algorithm like follows:

1. Detect the end of the currently transferred frame

2. Compute the start of the position of a new transferred frame (e.g compute offset)
3. Shift the Aligned frame with computed offset, fill the offset with zeros
4. Perform the logical OR operation to "merge" two frames together (currently transferred and the new one)
5. Transfer the rest of the frame with predefined offset
6. Continue with the first step if new frame is available

Chapter 4

Preliminary Results

4.1 Evaluation of Proposed Data Processing Unit

We described the architecture of the architecture for Data Processing Unit in Sec. 3.2.1. This section will describe achieved results for this kind of architecture.

As a demonstration of fast and simple implementation, we created the following instructions:

- The **Wormhole (W)** instruction is used for transformation of input request to the predefined output format. This unit doesn't perform any update at all, it is rather used for data output synchronization and transformation. The Wormhole module is implemented in VHDL because it is considered to be the part of the SDM Processor infrastructure.
- The **NetFlow (NF)** instruction is used for NetFlow aggregation – increase packet/byte counters, update flow start/end timestamps and compute logical OR of TCP flags. NetFlow module is implemented in both VHDL and C.
- The **NetFlow Extended (NFE)** instruction demonstrates an easy extension of existing C module implementation. Basic update functionality of this instruction is the same as described in NF. In addition, NFE stores the TCP flags of the first five packets. This additional information may become very useful for analysis of TCP handshake or for a detection of network attacks like DoS. The NetFlow Extended module is implemented in C with usage of the existing code from NF.
- The **TCP Flag Counters (TFC)** instruction shows the implementation of non-NetFlow kind of update. The instruction performs incrementation of counters of observed TCP flags. For example, one can see the number of ACK flags transmitted during the whole TCP connection. Information from this aggregate can be used to support advanced flow analysis [19]. This module is implemented in C++.
- The **Timestamp Diff (TD)** instruction shows another implementation of Non-NetFlow update. This instruction creates an aggregate record of inter-arrival times

of the first eleven packets of the flow. These times are computed with nanosecond precision. The module is implemented in C. Information from this aggregate can be used as network discriminators for flow-based classification [19] or for an identification of application protocol [20].

- The **CPD** instruction shows the most complex implementation of Non-NetFlow update. CPD stands for Change-Point Detection, which is an algorithm designed to detect an anomaly in the processed network flow. Detailed description of this method is out of scope of this work, more details can be found in [27, 3]. This method follows the equation (4.1) where S_{k-1} is the historical value of the measured statistic, m is a historical estimate of the measured statistic and c is an experimentally chosen coefficient to minimize the average detection delay. Current observed statistic N_k is computed as a ratio between the number of TCP SYN and ACK flags in the current flow.

$$S_k = max\{0, S_{k-1} + N_k - m - c\} \tag{4.1}$$

  Note that constants m and c are parts of the state record, which is stored in State Memory. To obtain the result of detection, the computed statistic S_k is compared to a threshold value to decide whether the network is under attack. This last step is left for the software part of the application.

To verify the designed Update part of the SDM Processor, we have implemented a prototype of the described architecture. We chose the board equipped with Virtex-7 H580T FPGA which is powerful enough for processing of 100 Gb/s traffic. Vivado HLS version 2013.2 was used for high level synthesis (C to VHDL). Xilinx ISE version 14.6 was used for VHDL to FPGA netlist synthesis. We have disabled all synthesis optimizations to avoid the register duplication and other modifications that can lead to higher frequency but also to higher resource usage. The resource usage of the instruction blocks is shown in Table 4.1. We can see that handmade NF instruction is better in the term of resource consumption but the HLS made instruction is also able to process all passed traffic, which makes the HLS usable for 100 *Gbps* networks.

Table 4.1: Resources of the instruction blocks

Instruction	**Registers**	**LUTs**	**Frequency [MHz]**
W – Wormhole	495	504	627.559
NF – NetFlow (handmade)	1754	325	425.134
NF – NetFlow (HLS)	1846	824	308.641
NFE – NetFlow Extended	2070	1113	308.641
TFC – TCP Flag Counters	0	1046	327.868
TD – Timestamp Diff	5199	2556	306.748
CPD – Change-Point Detection	5296	3919	305.815

Table 4.2 shows the total resource utilization of the whole SDM Update module. Each line of this table lists the instructions supported in the synthesized module. Frequency results from High Level Synthesis are good enough for processing the high-speed network traffic. We consider the minimal frequency of 200 MHz and the initialization interval of one as the requirements for 100 Gb/s traffic processing.

Table 4.2: Resources of SDM Update

Supported Instructions	**Registers**	**LUTs**	**BRAMs**	**Frequency [MHz]**
W,NF	3453	5024	24	294.772
W,NFE	4867	4399	24	281.897
W,TFC	3012	4636	24	296.002
W,TD	7074	6598	24	286.090
W,CPD	5997	6395	24	251.620
W,NF,TFC	5704	8107	48	285.531
W,NF,TD	9762	10360	48	275.820
W,NF,CPD	8685	11571	48	230.880
W,NF,TFC,TD	11965	12998	72	255.660
W,NF,TFC,CPD	12450	16153	72	225,279
W,NF,NFE,TFC,TD	16018	15969	96	238.792

BRAMs scale linearly with the number of instructions because the Reserve module (see Figure 3.1) uses BRAMs to store the reservation information. Note that all synthesized variants of the SDM Update module meet the frequency requirement for processing of 100 Gb/s traffic.

4.1.1 Conclusion of Data Processing Unit

We introduce the application specific processor with high level synthesized instructions. We use C/C++ language for instruction description to provide the easy way of development for non-HDL programmers.

We have performed frequency and FPGA resources evaluation for the powerful Virtex-7 H580T FPGA, which is large and fast enough to host a complex network processing hardware. Our SDM Update processor is ready to support the processing of $100\,Gb/s$ traffic for all presented instruction combinations.

Compared to the hand-written instructions, the high level synthesized instructions are slightly worse in terms of both FPGA resources consumption and frequency. However, the time needed for development is approximately half and the new instruction can be created by a user who doesn't have deep hardware knowledges. This is possible via high level synthesized description because it is using the wide known languages like C or C++. This brings the new possibility of network packet processing which can be implemented by experts in specific domains like security, etc.

The presented method of development enables faster implementation and verification of new modern monitoring methods that are based on the aggregation of flow records.

4.2 Evaluation of High-Speed Network Merger

We described the architecture of the High-Speed Network Merger in Sec. 3.3. Functionality of the implemented design was tested by functional verification. The referential model was written in System Verilog language and it was tested on many scenarios, including short packets only, fast TX and slow RX, slow TX and fast TX, etc. All defined tests were successfully met.

We have implemented and integrated the VHDL prototype into our development platform. This platform is equipped with two 40 *Gbps* ports which are switched to the second mode ($4 \times 10\,Gbps$) – we need to merge eight 10 *Gbps* streams. Tab. 4.3 shows detailed list of all resources for different configuration of the Effective Network Stream Merger. Parameters *Bus Width* and *Ports* don't have to be discussed. The *Align* parameter express the effectiveness property of all incoming streams. Therefore, value 0 means that all incoming flows are aligned and they can be directly used as FLUA data stream. Finally, value 1 means that all incoming flows are not aligned (e.g. FLU streams) and they have to be transferred to FLUA data stream (using the FLU2FLUA module). This transformation was described in previous section.

Table 4.3: FPGA resources used for the Effective Network Stream Merger; FPGA Virtex 7vx690tffg1157; Synthesis tool Xilinx XST 14.7

Bus Width	**Align**	**Ports**	**Registers**	**LUT**	**Frequency [MHz]**
256	0	2	946(0%)	1818(0%)	292.969
256	0	4	1500(0%)	3355(0%)	275.897
256	0	8	2078(0%)	5592(1%)	219.250
256	0	16	3898(0%)	8711(2%)	265.119
256	1	2	4177(0%)	6737(1%)	233.677
252	1	4	8090(0%)	10734(2%)	269.181
252	1	8	14990(1%)	20209(4%)	219.508
252	1	16	29848(3%)	37670(8%)	264.339
512	0	2	1980(0%)	6882(1%)	271.720
512	0	4	3956(0%)	8386(1%)	258.927
512	**0**	**8**	**4145(0%)**	**12969(2%)**	**206.527**
512	0	16	10570(1%)	19360(4%)	241.117
512	1	2	8307(0%)	17943(4%)	202.929
512	1	4	16226(1%)	28429(6%)	259.412
512	1	8	29440(3%)	52460(12%)	207.694
512	1	16	57968(6%)	102692(23%)	212.717

Tab. 4.3 also shows percentual usage of powerful Virtex 7vx690tffg1157 FPGA. Interesting configuration is marked in bold. As we can see, it doesn't consume huge amount of resources. Moreover, tab. 4.4 shows that maximal theoretical throughput via the output bus is enough to accommodate desired throughput of 80 $Gbps$. Notice, Maximal Throughput covers the situation where the communication line is used without wasting of the available place. Tab. 4.4 also shows good choice of bus width because all configurations with 256 bits are not able to meet desired throughput.

Table 4.4: Theoretical Output Throughput

Bus Width	**Align**	**Ports**	**Frequency [MHz]**	**Maximal Thr. [Gbps]**
256	0	2	292.969	75.00
256	0	4	275.897	70.63
256	0	8	219.250	56.13
256	0	16	265.119	67.87
256	1	2	233.677	59.82
252	1	4	269.181	68.91
252	1	8	219.508	56.19
252	1	16	264.339	67.67
512	0	2	271.720	139.12
512	0	4	258.927	132.57
512	**0**	**8**	**206.527**	**105.76**
512	0	16	241.117	123.45
512	1	2	202.929	103.90
512	1	4	259.412	132.82
512	1	8	207.694	106.34
512	1	16	212.717	108.91

Implemented design was also tested in a hardware environment. The implementation platform is the INVEA-TECH's COMBO-80G card [11] which is equipped with two 40 $Gbps$ Ethernet ports and Virtex7 VX690T-2 FPGA. The card is communicating via PCIe x8 gen3 slot which is able to transfer approximately 8 GB/s. There were tested two basic designs NIC [17] and HANIC [16]. We translated both designs, NIC met the frequency of 200 MHz and HANIC the frequency of 140MHz. Both designs were able to saturate PCIe x8 gen3 bus with constant data stream.

The NIC design at frequency of 200 MHz was able to transfer more than 60 $Gbps$ in FPGA firmware without packet loss. Faster transfers can't be tested because our testing environment is not able to generate more than 60 $Gbps$. The NIC design was also tested at frequency of 100 MHz. The result of this test shows that we are able to transfer minimally 43 $Gbps$. The transfer rate doubles at speed of 200 MHz. The Effective Network Stream Merger will be able to merge all incoming network streams. Measured data is shown in Fig. 4.2.

The red line is showing expected throughput inside the HANIC firmware. The orange

line is showing the expected throughput from the HANIC to the software. Finally, the black line is showing the measured throughput. Measured throughput is almost effective but there are some situations when the throughput is slightly lower than expected value. This inefficiency won't be problem at all because the High-Speed Network Stream Merger is designed for the frequency equal to 200 MHz. Therefore, the final design will be approximately 1.4× faster.

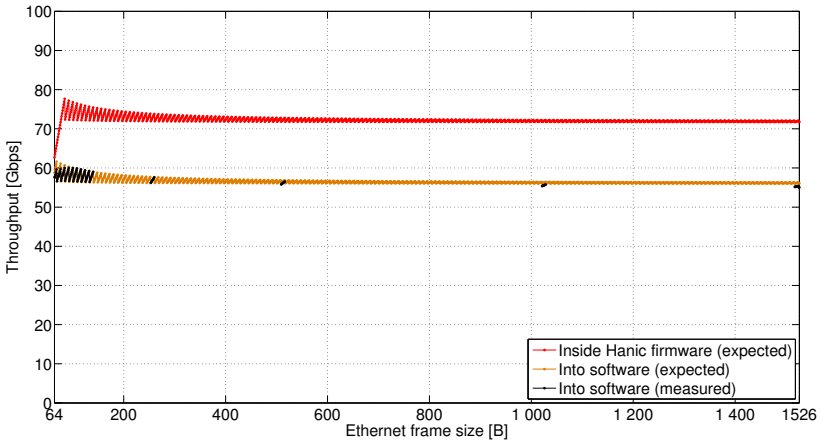


Figure 4.1: HANIC throughput of the High-Speed Merger; this graph shows the final throughput for the HANIC design which is running at frequency of 140 MHz

4.2.1 Conclusion of High-Speed Network Merger

In this section we have tested the architecture of effective high-speed network stream merger. We have performed frequency and FPGA resources evaluation of the hardware implementation for the Xilinx Virtex 7 VX960T-2 FPGA, which is large and fast enough to accommodate a complex networking hardware.

In our test case, the proposed architecture is able to merge eight 10 $Gbps$ network streams at frequency of 200 MHz. Therefore, our solution is able to merge all streams and saturate the output bus. Moreover, it is able to generate the effective output stream using the FLU protocol which was described in previous Sec. 3.3.1 which is describing theoretical requirements for this architecture.

Chapter 5

Conclusions

This report presents the actual approaches and state of the art in three basic groups of network operations. Chapter 1 introduces the basic motivation for the technology replacement with faster and more modern networking devices.

Chapter 2 introduces the background and related work for three basic groups of operations – Data Extraction, Classification and Data Processing (mainly the stateful processing). These groups of operations are very tightly bounded and they are very important for the final throughput of a networking device. We introduced novel approaches like Software Defined Monitoring [12], Query-Drive Measurement [13] or pipelined packet parser for FPGA based solutions [25]. It also discusses the importance of the HLS in the future development of networking devices.

Chapter 3 introduces our approach in the Data Processing and High-Speed Network Stream Merging. We introduced the novel architecture with general infrastructure. This infrastructure allows us to use the HLS approach in definition of user-defined processing blocks. The High-Speed Network Merging block is important entry point to the networking device. It allows merging of more high-speed network streams into the one high throughput stream. This approach enables us to save FPGA resources and reuse already created high-speed networking modules for 100 *Gbps* networks. Both proposed architectures were tested in the real environment.

Chapter 4 introduces our reached results for both proposed architectures – Data Processing and High-Speed Network Merging. Data Processing results were shown on few user-defined operations which can be used for complex non-NetFlow analysis. This section also presents our results for the High-Speed Network Stream Merger. It is able to merge all incoming data without any packet loss. Moreover, the design of the proposed architecture is scalable for higher throughputs and the merging process is as effective as possible. This fact was demonstrated on two designs – NIC [17] and HANIC [16]. The latest results shows the possibility of merging of ten 10 *Gbps* lines without any packet loss.

The following chapter describes future work and main points of my dissertation thesis in the area of high-speed computer networks with aim to the usage of the HLS.

Chapter 6

Proposed Doctoral Thesis

Title: *Methodology for Analysis of High-Speed Networks*

The planned aim of my dissertation thesis:

- Wider usage of the HLS approach in high-speed networks.
- Definition of general infrastructure which can be used for accommodation of user-defined modules.
- Definition of required architectures which enables to process traffic at speed of line (100 *Gbps* and above).
- We want to combine the HLS and modules written by hand to reach the maximal performance. This approach allows faster implementation of the design for domain oriented experts.

The author of the dissertation suggests to use the HLS approach in the following areas:

6.1 Parsing

We introduced this operation in Sec. 2.2.1 where we described and discuss the usage of the HLS in this group of network operation. Puš et al. [25] introduced the modern approach of a pipelined packet parser for FPGA. This parser has well defined general interface and it offers low latency and high throughput. However, the actual implementation doesn't contain any HLS support. We want to define a description, rules and algorithms which can be used for generation of such processing pipeline. Moreover, the author of the thesis wants to create a set of rules, algorithms and architectures to enable us an easy and fast implementation of parsing for new protocols.

For example, Virtual Extensible LAN (VXLAN) [8] is a network virtualization which is similar to classic VLAN [26]. However, it is not working on a data layer (L2) but it is using the UDP protocol for transfer of the network data. Therefore, infrastructure has dynamic

behavior which is very important for modern data centers. This approach requires higher utilization of the HLS for high-speed networks. The author of the paper also expects the introduction of new architecture for such defined parser.

6.2 Longest Prefix Match

This operation is very important during the classification process. The Longest Prefix Match (LPM) is very important for decomposition based methods. Matoušek et al. [18] have proposed a hardware architecture for efficient memory lookup in 100 *Gbps*. This novel approach is using modified *Tree Bitmap*(TBM) [7] with modified tree representation. Authors of the paper identify the most common decision tree structures.

Therefore, encoding of such decision trees can be optimized and we can reach the same result with less memory. However, authors are not talking about usage of the HLS which can be used for the description of proposed decision trees. The structure of the decision tree can be defined with usage of HLS. This definition can be used for the generation of the matching hardware architecture. Moreover, it can be combined with partial dynamic reconfiguration which brings more dynamic behavior.

6.3 Extension of the Classification Algorithms

We introduced basic approaches for decomposition based algorithms in Sec. 2.2.2. This approach is based on rules and pseudorules. Typically, small number of rules generates many pseudorules which need to be stored in the hardware. Many algorithms were developed to minimize the amount of pseudorules. Our approach is to extend this method and allow us to store more rules in the build-in memory.

6.4 Data Processing

In Sec. 2.2.3, we introduced basic approaches in the stateful data processing. We introduced our novel approach for processing of such data with usage of the HLS and Software Defined Monitoring. The SDM Processor was demonstrated with four different user-defined instructions which were described in C/C++ language. However, actual implementation doesn't support functionality which is useful for advanced statistical analysis.

For example, we can't run the analysis when some trigger is detected. Therefore, we plan to introduce novel architectures, algorithm and design rules to enable us such functionality. We need to study and discuss these requirements with mathematicians to make the architecture independent and general enough for variety of statistical and non-statistical traffic analysis.

Bibliography

[1] Balasaheb S. Agarkar and Uday V. Kulkarni. A novel technique for fast parallel packet classification. *International Journal of Computer Applications*, 76(4):18–25, August 2013.

[2] M. Attig and G. Brebner. 400 Gb/s programmable packet parsing on a single FPGA. In *Architectures for Networking and Communications Systems (ANCS), 2011 Seventh ACM/IEEE Symposium on*, pages 12–23, Oct 2011.

[3] Rudolf B. Blažek, H. Kim, B. Rozovskii, and A. Tartakovsky. A novel approach to detection of denialofservice attacks via adaptive sequential and batchsequential changepoint detection methods. In *Proc. 2nd IEEE Workshop on Systems, Man, and Cybernetics, West Point, NY*. 2001.

[4] Florian Braun, John Lockwood, and Marcel Waldvogel. Protocol wrappers for layered network packet processing in reconfigurable hardware. *IEEE Micro*, 22:66–74, 2002.

[5] T. Dedek, T. Martínek, and T. Marek. High level abstraction language as an alternative to embedded processors for internet packet processing in FPGA. In *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on*, pages 648–651, Aug 2007.

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

[7] Will Eatherton, George Varghese, and Zubin Dittia. Tree bitmap: Hardware/software ip lookups with incremental updates. *SIGCOMM Comput. Commun. Rev.*, 34(2):97–122, April 2004.

[8] M. Mahalingam et al. RFC 7348: Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks, August 2014.

[9] R. Giladi. *Network Processors: Architecture, Programming, and Implementation.* Systems on Silicon. Elsevier Science, 2008.

[10] IEEE 802.3 Ethernet Working Group. IEEE 802.3 Industry Connections Ethernet Bandwidth Assessment, 2012.

[11] INVEA-TECH. *COMBO-80G FPGA Card*. `https://www.invea.com/en/products-and-services/fpga-cards/combo-80g`, 2014. Accessed: 2014-10-02.

[12] Lukáš Kekely, Viktor Puš, and Jan Kořenek. Software defined monitoring of application protocols. In *INFOCOM, 2014 Proceedings IEEE*, pages 1725–1733, April 2014.

[13] Faisal Khan, Lihua Yuan, Chen-Nee Chuah, and Soheil Ghiasi. A programmable architecture for scalable and real-time network traffic measurements. In *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS '08, pages 109–118, New York, NY, USA, 2008. ACM.

[14] P. Kobiersky, J. Kořenek, and L. Polčák. Packet header analysis and field extraction for multigigabit networks. In *Design and Diagnostics of Electronic Circuits Systems, 2009. DDECS '09. 12th International Symposium on*, pages 96–101, April 2009.

[15] Maysam Lavasani, Larry Dennison, and Derek Chiou. Compiling high throughput network processors. In *Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays*, FPGA '12, pages 87–96, New York, NY, USA, 2012. ACM.

[16] Liberouter. *HANIC*. https://www.liberouter.org/technologies/hanic/, 2014. Accessed: 2014-10-02.

[17] Liberouter. *NetCOPE*. https://www.liberouter.org/technologies/netcope/, 2014. Accessed: 2014-10-02.

[18] J. Matoušek, M. Skačan, and J. Kořenek. Towards hardware architecture for memory efficient ipv4/ipv6 lookup in 100 gbps networks. In *Design and Diagnostics of Electronic Circuits Systems (DDECS), 2013 IEEE 16th International Symposium on*, pages 108–111, April 2013.

[19] Andrew W. Moore, Denis Zuev, and Michael L. Crogan. Discriminators for use in flow-based classification. Technical report, Department of Computer Science, Queen Mary University of London, 2005.

[20] Pavel Piskač and Jiří Novotný. Using of time characteristics in data flow for traffic classification. In Isabelle Chrisment, Alva Couch, Rmi Badonnel, and Martin Waldburger, editors, *Managing the Dynamics of Networks and Services*, volume 6734 of *Lecture Notes in Computer Science*, pages 173–176. Springer Berlin Heidelberg, 2011.

[21] J. Postel. RFC 791: Internet Protocol, September 1981.

[22] J. Postel. RFC 793: Transmission control protocol, September 1981.

[23] Viktor Puš and Pavel Benáček. Application specific processor with high level synthesized instructions. In *The 2014 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA '14, Monterey, CA, USA - February 26 - 28, 2014*, page 246, 2014.

[24] Viktor Puš. *Packet Classification Algorithms*. PhD thesis, FIT BUT, 2012.

[25] Viktor Puš, Lukáš Kekely, and Jan Kořenek. Design methodology of configurable high performance packet parser for FPGA, April 2014.

[26] IEEE. Computer society. *IEEE Standards for Local and Metropolitan Area Networks: Virtual Bridged Local Area Networks*. IEEE Standard. IEEE, 2003.

[27] A.G. Tartakovsky, A.S. Polunchenko, and G. Sokolov. Efficient computer network anomaly detection by changepoint detection methods. *Selected Topics in Signal Processing, IEEE Journal of*, 7(1):4–11, 2013.

Publications of the Author

[A.1] P. Benáček, H. Kubátová and V. Puš. *Architecture of Effective High-Speed Network Stream Merger.* Digital System Design (DSD), 17th Euromicro Conference on Digital System Design, pp 459–464, Verona, Italy, 2014.

[A.2] V. Puš and P. Benáček *Application specific processor with high level synthesized instructions.* Proceedings of the 2014 ACM/SIGDA international symposium on Field-programmable gate arrays, pp. 246–246, Monterey, CA, USA, 2014.

[A.3] L. Kekely, V. Puš, P. Benáček and J. Kořenek. *Trade-offs and progressive adoption of FPGA acceleration in network traffic monitoring.* Field Programmable Logic and Applications (FPL), 2014 24th International Conference, pp. 1–4, Munich, Germany, 2014.

[A.4] P. Benáček, T. Čejka, H. Kubátová and R. Blažek. *Change-Point Detection Method on 100 Gb/s Ethernet Interface.* ACM/IEEE Symposium on Architectures for Networking and Communications Systems, Marina del Rey, CA, USA, 2014.

[A.5] P. Benáček, T. Čejka, H. Kubátová, L. Kekely and R. Blažek. *FPGA Accelerated Change-Point Detection Method for 100 Gb/s Networks.* Doctoral Workshop on Mathematical and Engineering Methods in Computer Science, Telč, Czech Republic, 2014.

[A.6] P. Benáček. *Analýza rychlých síťových přenosů.* Česko-slovenský seminář, Milovy ve Žďárských vrších ,Česká republika, 2013.

[A.7] P. Benáček. *Architektura pro měření v reálném čase na vysokorychlostních sítích.* Česko-slovenský seminář, pp. 45–50, Teplá, Česká republika, 2013.

[A.8] P. Benáček. *Real-time Network Measurement for High-Speed Networks.* The 1st Embedded Systems Workshop (ESW), Temešvár, Czech Republic, 2013.

Other publications:

[A.9] P. Benáček and M. Novotný. *Implementing Brute-Force Attack on PRESENT Cipher*. Digital System Design (DSD), 13th Euromicro Conference on Digital System Design, pp 51–52, Linz, Austria, 2010.

[A.10] P. Benáček, T. Čejka, Š. Friedl and R. Krejčí. *COMET - COMBO Ethernet Tester*. Annual report, CESNET z.s.p.o., Prague, Czech Republic, 2013. Available at:`http://www.cesnet.cz/wp-content/uploads/2013/03/comet.pdf`

[A.11] P. Benáček *Ethernet tester for high-speed networks*. Diploma thesis, Czech Technical University in Prague, Faculty of Information Technology, 2014.