BRNO UNIVERSITY OF TECHNOLOGY

Faculty of Information Technology

# PHD THESIS

Brno, 2017 Ing. Lukáš Kekely

**BRNO UNIVERSITY OF TECHNOLOGY**
**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**

**FACULTY OF INFORMATION TECHNOLOGY**
**FAKULTA INFORMAČNÍCH TECHNOLOGIÍ**

**DEPARTMENT OF COMPUTER SYSTEMS**
**ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ**


# SOFTWARE-CONTROLLED NETWORK TRAFFIC MONITORING

**SOFTWAROVĚ ŘÍZENÉ MONITOROVÁNÍ SÍŤOVÉHO PROVOZU**

**PHD THESIS**
**DISERTAČNÍ PRÁCE**


**AUTHOR** — **Ing. LUKÁŠ KEKELY**
**AUTOR PRÁCE**

**SUPERVISOR** — **Ing. JAN KOŘENEK, Ph.D.**
**ŠKOLITEL**



**BRNO 2017**

# Abstract

This dissertation thesis deals with the design of a novel hardware acceleration, software controlled (defined) concept for high-speed computer networks. The main goal is to propose general, flexible and easy to use acceleration platform for various network security and monitoring applications suitable for deployment in real 100 Gbps and faster networks. The thesis starts with the survey of the current state of the art in network monitoring, security and accelerated high-speed traffic processing. Based on the survey, a brand-new concept called Software Defined Monitoring (SDM) is formulated and proposed. A key feature of the concept lies in hardware accelerated, application specific (controlled), flow based, informed reduction and distribution of captured network traffic. This brings high-speed hardware processing coupled with flexible software control, which together leads to an easy creation of various complex high-performance network applications. Further optimizations and enhancements of the main SDM concept and its selected components are also explored resulting in creation of unique and novel designs of generally usable FPGA architecture of modular packet header parser and cuckoo hash based high-throughput packet classification engine. Finally, high-speed SDM prototype using FPGA acceleration network interface card is created and thoroughly evaluated under real network conditions. Achievable performance improvements in several chosen monitoring and security use case scenarios are measured and shown. The SDM prototype is also deployed in production monitoring of real backbone network by Cesnet association and has been commercialized by Netcope Technologies.



# Keywords

FPGA, hardware acceleration, SDN, Software Defined Networking, monitoring, security, high-speed networks


# Reference


KEKELY, Lukáš. *Software-Controlled Network Traffic Monitoring.* Brno, 2017. PhD thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Jan Kořenek, Ph.D.

# Abstrakt

Tato disertační práce se zabývá návrhem nového způsobu softwarově řízené (definované) hardwarové akcelerace pro moderní vysokorychlostní počítačové sítě. Hlavním cílem práce je formulace obecného, flexibilního a jednoduše použitelného konceptu akcelerace použitelného pro různé bezpečnostní a monitorovací aplikace, který by umožnil jejich reálné nasazení ve 100 Gb/s a rychlejších sítích. Disertační práce začíná rozborem aktuálního stavu poznání v oborech síťového monitorování, bezpečnosti a způsobů akcelerace zpracování vysokorychlostních síťových dat. Na základě tohoto rozboru je formulován a navržen zcela nový koncept s názvem Softwarově definované monitorování (SDM). Klíčová funkcionalita uvedeného konceptu je postavená na hardwarově akcelerované, aplikačně specifické (řízené), na tocích založené, informované redukci a distribuci zachycených síťových dat. Toto je zajištěno spojením vysokorychlostního hardwarového zpracování s flexibilním softwarovým řízením, které tak společně umožňují jednoduchou tvorbu různých komplexních a vysoce výkonných síťových aplikací. Pokročilé optimalizace a vylepšení základního SDM konceptu a jeho vybraných komponent jsou v práci též zkoumány, což vede k návrhu zcela unikátní a obecně použitelné FPGA architektury modulárního analyzátoru hlaviček paketů a vysoce výkonného klasifikátoru paketů založeného na kukaččím hashovaní. Nakonec je vytvořen vysokorychlostní SDM prototyp postavený nad FPGA akcelerační síťovou kartou, který je podrobně ověřen v podmínkách nasazení do reálných sítí. Jsou změřeny a diskutovány dosažitelné zlepšení výkonností v několika vybraných monitorovacích a bezpečnostních případech užití. Vytvořený SDM prototyp je rovněž nasazen v produkčním monitorování reálné páteřní sítě sdružení Cesnet a byl komercializován společností Netcope Technologies.



# Klíčová slova

FPGA, hardwarová akcelerace, SDN, softwarově definované zpracování síťového provozu, monitorování, bezpečnost, vysokorychlostní sítě


# Citace

KEKELY, Lukáš. *Softwarově řízené monitorování síťového provozu.* Brno, 2017. Disertační práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Školitel Ing. Jan Kořenek, Ph.D.

# Software-Controlled Network Traffic Monitoring

## Prohlášení

Prohlašuji, že jsem tuto práci vypracoval samostatně pod vedením Ing. Jana Kořenka, Ph.D. a školitele specialisty Ing. Viktora Puše, Ph.D. Nápomocní mi též byli kolegové ze sdružení Cesnet. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Lukáš Kekely
19. května 2017

## Poděkování

Chtěl bych poděkovat hlavně vedoucímu práce Ing. Janovi Kořenkovi, Ph.D. za jeho odbornou pomoc a rady při tvorbě této práce. Chtěl bych též poděkovat lidem ze sdružení Cesnet za spolupráci a poskytnuté informace, zejména pak Ing. Viktorovi Pušovi, Ph.D. za jeho odbornou pomoc.

# Contents


**1 Introduction** **7**
- 1.1 Research Area and Objectives . . . 8
- 1.2 Thesis Outline . . . 9

**2 Current State of the Art** **10**
- 2.1 Network Monitoring . . . 10
- 2.2 Network Security . . . 13
- 2.3 Software-Defined Networking . . . 14
- 2.4 High-Speed Network Traffic Processing . . . 16
- 2.5 Related Work . . . 18

**3 Research Summary** **20**
- 3.1 Core Ideas . . . 20
- 3.2 Research Process . . . 22
- 3.3 Papers . . . 23
- 3.4 List of Publications . . . 28

**4 Discussion and Conclusions** **30**
- 4.1 Results . . . 30
- 4.2 Conclusions . . . 35
- 4.3 Deployment and Usage . . . 35
- 4.4 Future Work . . . 36

**Bibliography** **37**

**A Included Papers** **41**
- A.1 Paper 1 . . . 42
- A.2 Paper 2 . . . 52
- A.3 Paper 3 . . . 57
- A.4 Paper 4 . . . 70
- A.5 Paper 5 . . . 77
- A.6 Paper 6 . . . 82

# Chapter 1

# Introduction

The most characteristic feature of the modern age is a constant progression in all branches of human endeavors. One of the fastest advancing fields is information (computation) technology, especially network communications. Currently, computer networks have shifted from being just a cheap and fast way of communication to become a platform for provisioning of a wide variety of other services (trade, advertisement, games, social media, multimedia ...).

Even though computer networks currently have a huge impact on everyday life, their influence keeps on rising implacably. A constant growth is apparent in the number of networking capable devices, the number of provided services, the number of active users and time they spend online every day. Apart from increasing user base size, the sheer amount of transferred data during each communication is also rising. These trends lead to clearly visible exponential growth in network traffic volume. In the last 5 years the volume of network traffic has grown 12 times [18] and a similar rate of growth is even expected to persist in years to come. The main consequence of the described growth is a need for more powerful and faster network infrastructure. Therefore, a shift from throughputs of 1 Gbps and 10 Gbps towards 40 Gbps and 100 Gbps is currently apparent in high-speed networks. Demands and preparations of even faster networking protocol standards are common (e.g. 400 Gbps Ethernet standardization [19] or terabit solutions requests by data centers).

Network infrastructure is composed of not only the basic functional devices that perform data transfers and traffic routing, but it should also contain special devices ensuring the security of the network itself and monitoring of its state. Furthermore, their performance cannot fall behind the rest of the network infrastructure. So, throughputs of monitoring and security devices (applications) must be rising at the similar rate as networks are speeding up. Therefore, the main goal of this thesis is to design a suitable platform that enables the creation of various network monitoring and security applications with sufficient performance for practical deployment in high-speed networks operating at 100 Gbps and more.

A common approach to the already described need for faster devices lies in the utilization of hardware accelerated techniques. The main advantage of hardware acceleration (compared to purely software approach) is in the specialization of the hardware architecture based on the specific needs of the task at hand. This enables for tailored utilization of various parallel and pipelined data processing approaches that can lead to notable increases in overall performance of designed devices.

On the other hand, hardware acceleration utilization can also bring several problems. Mainly, high specialization of hardware usually means a significant loss in its flexibility, what can be very troublesome in the ever-changing environment of networking protocols. Another significant issue is that many security and monitoring tasks (applications) are

too complex for practical hardware realization. For these kinds of tasks, an integration of software and hardware elements into a single system is more suitable. Such system can then have higher performance enabled by hardware acceleration of time-critical parts and can be also more flexible and easier to realize because of a software implementation of more complex parts. In computer networks, intelligent hardware and software integration is well utilized by Software-Defined Networking (SDN) [35]. The key idea defined by SDN is the division of the network functionality into intelligent software applications controlling relatively simple but powerful hardware devices.

The SDN ideas can be also useful for the area of network monitoring and security. Using current common approaches, it is very difficult to create practical high-speed realizations of anomaly and threat detection or deep packet inspection at the level of application protocols. Because these systems are relatively complex, their hardware realizations are unbearably complicated and require a lot of resources and development time. Software solutions, on the other hand, are much simpler, but their performance without traffic sampling is usually barely sufficient even for 10 Gbps networks and certainly not enough for 100 Gbps [48]. But, it should be possible to notably increase the poor throughput of software solutions by creation and utilization of an appropriate hardware acceleration platform. The performance could be significantly improved by software controlled hardware accelerated preprocessing of network packets that enable applications to directly select the level of information details retained by hardware and to distribute the software workload. So, the hardware platform would implement means for various kinds of network traffic preprocessing and division (routing) of captured packets into multiple CPU cores, which could be controlled on-the-fly from the software according to needs of running applications. Therefore, software applications would gain the ability to massively reduce the volume of traffic that they need to process and consequently achieve much higher throughputs.

## 1.1 Research Area and Objectives

As already indicated in the motivation, the main area of research presented in this thesis is design, realization and effectivity assessment of a novel software controlled hardware acceleration concept specialized for simple creation of network monitoring and security applications with throughputs sufficient for high-speed networks (100 Gbps and more). All the research steps are performed with emphasis on achievable performance improvements of various applications when deployed with the designed acceleration concept in the real networks. To achieve the desired performance improvements, this research aims at the utilization of various forms of controlled accelerated reduction of incoming traffic volume coupled with an intelligent division of application workload between multiple cores of modern CPUs. This way, applications should be able to effectively reduce the amount of processed uninteresting bulk traffic, what in turn increases their performance or leave them more CPU time for further (deeper) analysis of the most important parts of the data.

Based on the mentioned research ambitions in the area of network monitoring and security acceleration, the following working hypothesis is formulated for this thesis:

> *Utilization of appropriate concept of software (application) controlled hardware accelerated reduction and distribution (between processor's cores) of real network traffic will, compared to existing concepts, lead to considerable increases in total performance of selected network monitoring and security applications and/or better achievable quality (accuracy and detail) of information obtained by these applications.*

The main research objective is to design and consequently assess a suitable acceleration concept that is flexible enough to enable acceleration of not just one specific application, but many different ones.

The main objective of this thesis can be divided into several consecutive sub-goals:

1. Proposal of an appropriate concept of software controlled hardware acceleration of network monitoring and security applications.
2. Assessment of designed concept feasibility based on observed characteristics of traffic from real high-speed networks.
3. Design and creation of hardware architecture, that implements the proposed concept on the FPGA acceleration network interface cards with performance sufficient for 100 Gbps networks or faster.
4. Experimental evaluation of achievable performance and speed-up of selected network monitoring and security applications when utilizing the proposed acceleration concept in the real high-speed networks.
5. Analysis, design, and implementation of extensions to the main concept that would enhance achievable application speed-up and/or enlarge the set of accelerable tasks.
6. Detailed analysis of general approaches utilized in selected subcomponents of the designed hardware architecture with the aim to propose more effective FPGA implementations.

## 1.2 Thesis Outline

This thesis is written as a collection of papers. Research contribution is, therefore, presented by several selected peer-reviewed research papers and a journal article, all in their original format. These texts are provided at the end of the thesis as appendix A. The following text of the thesis itself is organized into three chapters. Chapter 2 provides a brief summary of the relevant state of the art. It includes basic principles of network monitoring and security, an introduction to Software-Defined Networking (SDN), currently used approaches of traffic processing acceleration in high-speed networks and discussion of relevant related works. Chapter 3 describes the conducted research and its contribution, the chapter also contains an overview of included research papers from appendix A. Finally, chapter 4 summarizes and discusses the conclusions of the thesis, it also proposes possible directions of future research on the topic.

# Chapter 2

# Current State of the Art

## 2.1 Network Monitoring

Rapid growth of computer networks utilization leads to major rising of network infrastructure complexity. Furthermore, a huge diversity of currently used network services results in a substantial number of different protocols and policies that must be supported. The overall complexity of networks is also negatively influenced by the dynamic character of their topology (e. g. one mobile device can connect from different points). To effectively manage such complicated systems, it is vital for administrators to be able to acquire precise information about actual state and traffic in controlled networks. Appropriate approaches how to acquire such information belong to the fields of network monitoring (alternatively called network measurement) and network traffic analysis. Description of basic principles of these fields presented in the following text of this section is based on information from [4, 45, 46, 28, 9].

Network monitoring and traffic analysis is, in general, based on periodic probing of devices states and/or collecting of data about ongoing communications. The main goal is to precisely identify what is happening on the network at any given time. Such information can be subsequently used by administrators to optimize network functionality and performance. Two basic types of data can be acquired from network monitoring. The first group consists of real-time network parameters like characteristics of ongoing communications or current volumes of traffic processed by individual devices. These parameters can be used to detect various network anomalies, for example, device failures or targeted attacks. The second group of information represents long-term statistics acquired by storing and aggregation of periodically measured parameters. These statistics show trends in network usage and are useful for proper planning of future infrastructure upgrades.

The research scope of this thesis is focused on monitoring of ongoing network communications that can be performed in several ways. Monitoring of communication parameters can be realized directly on devices that are already part of the main network infrastructure or by additional dedicated monitoring devices. Usage of dedicated devices usually enables a collection of more precise monitoring data and does not endanger the overall performance of the network by the introduction of additional tasks for infrastructure devices. Another division of network monitoring techniques is defined by measurement activeness. In other words, if they need to actively inject packets into the network or rely only on a passive measurement of already existing network traffic. Large scale usage of active approaches is not generally preferred, as they can introduce negative influences on the measured behavior of the monitored network.

By passive monitoring techniques various information about ongoing network communications are acquired, for example, transferred data volume, pairs of communicating entities (devices), commonly used protocols and services. The most basic and common tools of passive monitoring are packet capturing programs like wireshark and tcpdump. Also, many networking devices by various vendors implement support of MIB database and SNMP [3]. Figure 2.1 shows an example of possible passive monitoring deployment. Monitoring probe (sometimes called metering or measurement point) is looking at communications between two networks (illustrated as clouds). It analyzes transferred packets and extracts some statistical information from them.

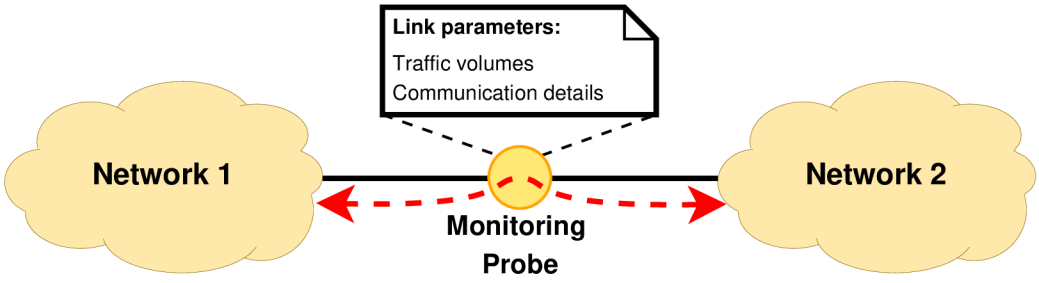


Figure 2.1: Passive network monitoring example.

One of the main challenges for passive monitoring is the sheer amount of data that needs to be processed. In high-speed networks, each link usually transfers millions of packets containing gigabits of data per second. Capturing and storing whole data or even basic information about all individual packets is unbearably storage intensive. To significantly reduce the volume of stored data, some information aggregation over groups of packets must be used. Most often, packets are grouped into network flows and only statistical information about these flows (called flow records) are stored.

Network flow [42] is defined as a sequence of packets that have same key features and are transferred through observed point in the network during a given time interval. The key features are usually selected fields from protocol headers and therefore, values of these fields uniquely identify a single flow for each packet. In current computer networks, which are mostly based on TCP/IP stack [6], flows are identified by fields from common protocol headers of network (IPv4 [40] or IPv6 [12]) and transport (UDP [39] or TCP [41]) layers. Specifically, five-tuple composed of source and destination IP address, source and destination port number and transport layer protocol number is often used. So, each network flow represents one direction of communication between two specific devices using given service. In flow-based monitoring, each flow record, apart from values of identifying key features, also stores additional statistical information aggregated from all corresponding packets. Stored values usually include at least characteristics of flow size (numbers of packets and bytes) and timing (start and end timestamp, or duration). Recently, mentioned basic characteristics are very often supplemented by various additional information obtained from application layer protocols or by deep packet inspection (e. g. pattern matching).

Passive flow-based monitoring of network is commonly realized using an architecture similar to one illustrated in figure 2.2. Probes (also called flow exporters) are placed in several places in the network in order to capture packets, extract and aggregate interesting information from them into flow records. Using specialized communication protocol (red arrows), flow records are exported from probes to a centralized collector. The collector receives and stores flow data from all probes, building a database for further analysis.

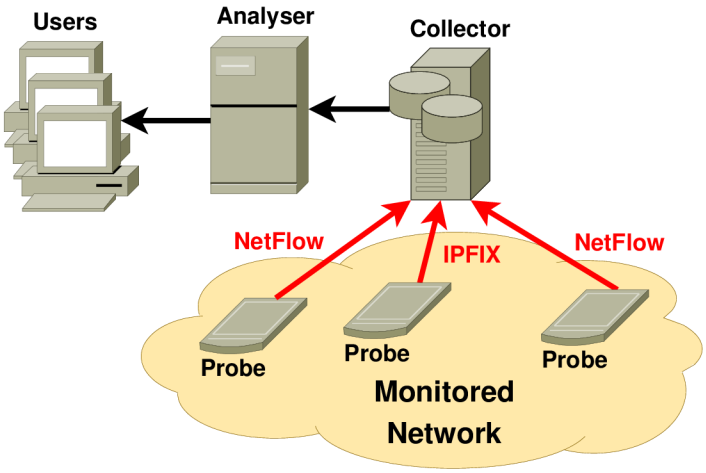


Figure 2.2: Architecture of flow based monitoring system.

Finally, the analyser presents collected data to users (network administrators) and can also perform automatic or semi-automatic advanced analyses to discover anomalies and other interesting features. From the point of view of this thesis, probes and communication between probes and collector are the most interesting parts.

Monitoring probes are usually realized as dedicated hardware devices or commodity servers with specialized acceleration network interface cards. Hardware resources must provide sufficient performance to capture and process as much network traffic as possible, ideally without any packet loss. Processing of each packet starts with analysis and extraction of flow identifying and other interesting values. Then, corresponding flow record is found and updated in (or added into) a table of active flows (called flow cache) managed by the probe. To correctly aggregate information in flow records, packets of a single flow must be, most of the time, processed in order of arrival. This requirement constraints division of incoming packets processing between multiple independet CPU cores. Apart from flow records updates and creation by processed packets, the probe must also manage periodic removal and export of finalized records from its flow cache. A flow is considered as finalized in the following four cases [8]: (1) implicit detection of the last packet, e. g. set SYN or RST flag in TCP header [41]; (2) expiration of active timeout, i. e. the flow is lasting too long (usually in the order of minutes); (3) expiration of inactive timeout, i. e. no packets of the flow have been captured recently (usually in the order of seconds); (4) forced artificial finalization when flow cache becomes full. Flow records can be stored for a while after finalization to be exported from the probe in bulk.

The most common protocol used for flow record export is currently NetFlow in versions 5 or newly 9 [8]. NetFlow is a proprietary protocol developed by Cisco, but it is considered an industry standard because of its wide adoption. It defines rather fixed flow record format and furthermore, in version 5 supports only IPv4 combined with TCP or UDP (no IPv6, IPX nor VLAN). To solve the shortcomings of NetFlow, IPFIX (IP Flow Information Export) [7] protocol was defined and slowly employed. IPFIX is developed by IETF as vendor independent standard. It basically extends NetFlow version 9 towards the support of considerably more flexible flow record structure by enabling definitions of custom information and data models for exported data. Furthermore, IPFIX standard also defines requirements for monitoring architecture (similar to Figure 2.2) and functionality implemented by each probe.

The currently ongoing trend is towards creation of richer flow records, often by inclusion of values from the application layer protocol headers, such as HTTP, DNS etc. It seems, that the ability to analyze application layer is crucial for improvement of the quality of network monitoring, because more and more of the network functionality is being shifted up in the protocol stack. While introduction of IPFIX solves the task of exporting/transmitting the additional application layer data in flow records, there remains the issue of obtaining them. This process inevitably requires additional computational resources. Implementation of the application level flow monitoring with a commodity CPU is certainly possible, yet its throughput is limited mainly by the performance of the processor cores [17]. On the other hand, many proposed ASICs and FPGAs approaches offer much better possibilities in terms of throughput. However, a fixed solely hardware implementations face the flexibility issues, since the evolving nature of networks implies the need for fast changes of the monitoring process, quickly making fixed hardware devices obsolete.

## 2.2 Network Security

Text of this section is mainly based on information from [33, 24, 5, 23, 11]. Connecting a device or a network to the internet creates a physical link towards thousands of other unknown networks and millions of users. On one side, this connection provides easy access to all kinds of useful services and information. But on the other hand, it also creates an access possibility in the opposite direction – from other users. Furthermore, some of the connected devices usually contain data and services that should not be accessible to everyone. Therefore, the existence of reliable techniques for assurance of access limitation and information security is necessary for networking. Precisely these techniques and challenges from these areas are studied by the field of network security.

Network security is more and more important as the significance of computer networks is raising. It is also common that administrators spent more time securing networks than configuring and managing their core operations. The main objective of network security is two-fold. First, it is information security, where confidential data must be protected from unauthorized access and modification. This protection needs to be ensured on devices inside the network as well as in legitimate traffic leaving the network (e. g. by encryption). Second, it is a protection of network, its infrastructure and devices from malicious attacks or misuse by unauthorized users. The ability of early detection of potential threats and ongoing attacks plays a crucial role.

For adequate detection of potential threats, common network attack methods and their signatures must be recognized. As network protection tools are getting better, malicious programs are also advancing and new increasingly sophisticated attack vectors are used. Individual types of attacks can differ from each other in purpose, complexity, and hazardousness for various devices. Common types of malicious activity used by attackers include: (1) scanning of active devices in the network to pinpoint potential targets for and forms of subsequent main attack; (2) denial of service for legitimate users by overloading the hosting device (or network infrastructure itself) with massive amounts of artificially generated communications; (3) malware programs designed to automatically spread between network devices using known vulnerabilities and perform various malicious activities on them (e. g. information theft, a base for further attacks); (4) determined penetration of network defenses to gain unauthorized access to various systems and databases. A substantial number of network attacks of mentioned types are well known and therefore, can be recognized by their characteristic communications patterns. On the other hand, there are always new

attacks with so far unknown behavior. These must be usually uncovered based only on observed statistical anomalies and deviations in general network traffic parameters. Both types of signs, specific communication patterns, and statistical anomalies, can be for some groups of attacks detected from network monitoring data.

Detection of malicious activity is usually only the first step towards robust network security. A key part of this robustness lies in a separation of the inner network (often rather trusted) from unknown or uncontrolled surroundings – i. e. establishing of a network perimeter. Well-made defenses of the perimeter can consist of multiple layers using diverse types of security methods like traffic monitoring, threat detection, user authentication or access limitation. These kinds of perimeter defenses notably reduce the risk of successful external attacks but do not provide any protection against internal threats (e. g. attacker with physical access). Therefore, it is also advised to somehow secure inner parts of the network, what is usually achieved by hierarchical division of the network into smaller separated subnets and by securing individual devices by antivirus programs and frequent updates.

Network infrastructure devices of various vendors directly provide some limited means to aid network security. But usually, specific dedicated devices and applications must be added to ensure sufficient defense of a network. The most common ones are firewalls, intrusion detection systems (IDS) and intrusion prevention systems (IPS). Firewall is a device or application guarding the access to the network by filtering entering or leaving communications (packets) based on a set of rules. There are several kinds of firewalls providing different types of filtering (e. g. static, stateful, proxy). Next, the main job of IDS is a detection of unwanted, potentially malicious events in network traffic and informing administrators about them. Generally, IDS directly process captured packets and try to identify predefined patterns (signatures) or statistical anomalies (sometimes separated as Anomaly Detection Systems, ADS) in them. Each signature is usually valid only for a specific type of traffic, therefore, not all signatures need to be searched in all captured packets. Finally, IPS extends detection of threats by an automatic response to them and can be sometimes viewed as a coupling of IDS with firewall functionality.

## 2.3 Software-Defined Networking

The content of this section is based on information acquired from [35]. Software Defined Networking (SDN) is a novel concept of the architecture of computer networks, which is currently gaining popularity. It has been created and is also further developed as a common effort of multiple commercial subjects joined under Open Networking Foundation (ONF). The ONF is a non-profit industrial consortium dedicated mainly to development and standardization of key elements of SDN architecture. One of the first achievements of ONF is the creation of OpenFlow [37] protocol standard as the first unified communication interface for SDN-enabled devices.

The key motivation behind the creation of SDN is the need for a new networking paradigm that is fueled by an apparent shift in nature of computer network utilization towards more dynamic patterns. Conventional network architecture models are mainly static and optimized to effectively support the client-server type of communications. On the other hand, more and more network services do not employ this type of communication. Some of the key computing trends driving this shift in nature of network communication include changes in usage (traffic) patterns, the spread of mobile devices, and the rise of data volumes (Big Data). Furthermore, traditional architectures of network infrastructure more often lead to problems severely limiting the effective utilization of today's networks. The

main limitations are for example network complexity leading to its stasis, inconsistency in sets of policies, poor scaling opportunities, and strong vendor dependence. The mentioned problems are taken into consideration when defining individual SDN components.

The main idea behind network infrastructure design in SDN concept lies in a strict decoupling of network intelligence (control plane) from functional elements performing traffic forwarding itself (data plane). Furthermore, the network control plane is logically centralized and directly programmable. Described idea of decoupling in SDN can be compared to the architecture of current personal computers, where CPU cores (data plane) performs instructions defined by executed programs (intelligence). Decoupling of control and data planes enables definition of multiple abstraction layers over the functional network infrastructure. Thanks to such abstractions, configuration, and automatization of various network operations are much easier to achieve, what in turn notably simplify the creation of more scalable and flexible network infrastructures.

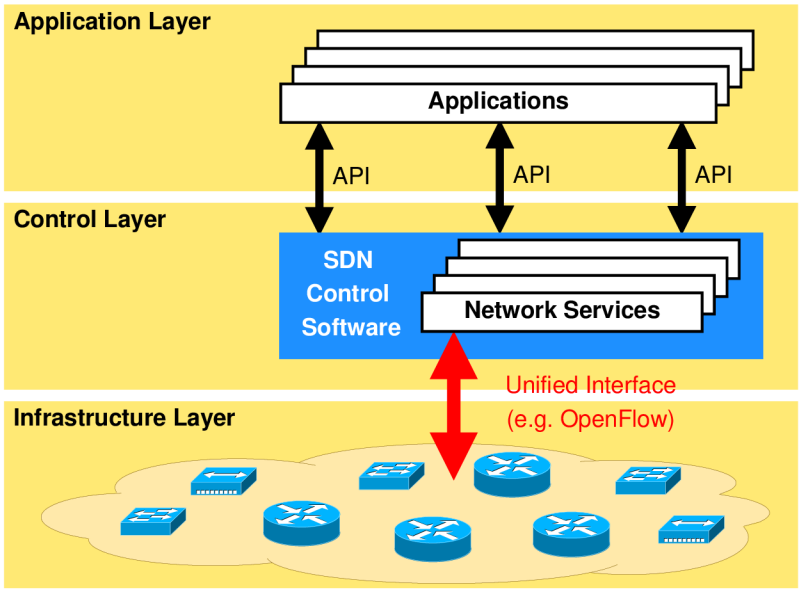


Figure 2.3: Basic overview of Software-Defined Networking architecture.

From a logical point of view, the architecture of SDN is divided into a few layers illustrated by Figure 2.3. On top of actual physical devices of network infrastructure (bottom part of the figure), there is an abstract control layer. In the control layer, network intelligence is logically centralized in software-based SDN controllers that maintain and update the global view of managed network. As a result of such centralized control layer, applications (in application layer) can view the whole network as one large switch. SDN thus enables management of the whole network from a single logical point in a vendor (device) independent manner. Described shielding from vendor-specific characteristics of infrastructure devices leads to considerably simpler design and management of networks. On the other hand, the definition of unified control interface in SDN enables simplification of infrastructure devices as well – their sole purpose now is just to receive and execute instructions from SDN controllers.

Apart from abstraction towards physical network devices, SDN architecture defines a set of application interfaces (APIs). These APIs make it possible to implement common network services like routing, multicast, access control, quality of service, security and all

types of policy management, based on the specific requirements imposed. Furthermore, thanks to centralization, the deployment, and management of these services in a consistent manner throughout the whole network is considerably simplified. Also, unified APIs between SDN controllers and user applications enable utilization of network resources without being tied to specific details of their implementation. SDN, therefore, views the networks as systems providing various capabilities that can be utilized in a customized (optimized) way for individual applications.

ONF activities around SDN have, till this day, lead to creation and improvement of OpenFlow protocol. OpenFlow [37] represents the first standardized communication interface between control and infrastructure layers of SDN architecture. It defines basic primitives for remote control and configuration of routing performed by physical network devices. To identify and control network traffic, OpenFlow uses an approach based on network flows that are described in the previous text (section 2.1). The approach utilizes a set of pre-defined static and dynamic matching rules specifying processing (forwarding) of individual flows or their groups. These rules are configured by SDN control software and can be based on various network parameters. Programmability on per-flow basis provides extremely granular control of network behavior, enabling precise response of the network to real-time changes at application, user or session levels.

## 2.4 High-Speed Network Traffic Processing

Network monitoring and security applications are all based on some kind of captured traffic (packets) processing that extracts interesting information of various kinds for further use. But, with the already mentioned rapid increase of network speeds, this traffic processing becomes a critical performance bottleneck for these applications. Performance requirements of traffic processing in the worst-case for different network speeds are illustrated in table 2.1. As one can see, at speeds of 10 Gbps and more, tens or even hundreds of millions of packets must be captured and processed every second. So, only a few nanoseconds can be spent on each packet and that is precisely why the performance of exactly this operation is very often hardware accelerated. In current high-speed networks operating at 10 and 40 Gbps, two main models of traffic processing acceleration are prevalent – software application aided by hardware acceleration network interface cards or specific fully hardware accelerated device.

| **Link speed** | **Packets per second** | **Time to process a packet** |
|---|---:|---:|
| 1 Gbps | 1 488 095 | 672.00 ns |
| 10 Gbps | 14 880 950 | 67.20 ns |
| 40 Gbps | 59 523 800 | 16.80 ns |
| 100 Gbps | 148 809 500 | 6.72 ns |
| 400 Gbps | 595 238 000 | 1.68 ns |

Table 2.1: Network traffic performance requirements at different speeds.

Hardware accelerated network interface cards (NICs) are usually connected to a host computer (server) over PCI Express bus and accelerate only traffic capturing, its very basic preprocessing (L1 decapsulation, FCS control, timestamping . . . ) and transfer to the host main memory. All main packet processing is then performed by monitoring and security applications fully running in CPUs of the host computer. The main advantage of such approach is its huge flexibility as hardware accelerated is only the most general part of

data processing. All application specific tasks are left for software, which can be changed and modified considerably easier compared to hardware. The main disadvantage, here, is generally worse performance as the whole volume of captured traffic must be delivered to and processed by software. For 100 Gbps and faster networks, this approach hits two major bottlenecks: insufficient performance of CPUs and limited throughput of PCI Express bus.

Many hardware accelerated interface cards has been introduced in recent years, for example: Hanic platform on cards from COMBO family [44, 27], NetFPGA SUME accelerated NIC [52], Endace DAG data capture cards [14] or various Napatech NACs products [31]. These platforms support wire-speed capture and processing of network traffic from connected high-speed links and transfers of the captured data over PCI Express using direct memory access (DMA). One of the primary available features is the ability to intelligently distribute transferred data into multiple software channels in a round robin or flow-aware manner. This division enables somehow independent processing of data on each channel by dedicated CPU core with limited need of their synchronization, thus rather effective division of workload is achieved. Other features supported by some include sampling (random or flow based), packet cropping, precise timestamping and basic filtering (classification). Therefore, most of the application specific traffic processing is left to be performed by software applications.

The second acceleration model is fully hardware accelerated system, which is based on utilization of hardware device (architecture) dedicated for acceleration of a single specialized task. The main advantage of highly specialized hardware is rather high performance as most of the network traffic processing is accelerated. But, such narrow specialization of hardware dramatically reduces its flexibility and increases development cost, as different tasks require different hardware devices or introduce considerable changes into hardware architecture. Furthermore, more complex tasks, e. g. analysis of application protocols or threat signatures detection can be too demanding or even impractical for hardware realization. Because generally, design and implementation of some task in hardware are severalfold more difficult than in software.

To represent fully accelerated approach, FlowMon probe [13] can be selected as an example of monitoring system. FlowMon probe utilizes NetCOPE development platform [26] on COMBO cards or NetFPGA cards [30]. It is a completely hardware accelerated NetFlow probe for 1 and 10 Gbps networks, where implementation of all packet processing and flow cache management operations is realized directly in FPGA firmware. As examples of security systems, fully accelerated regular expression (string matching) platforms like [29, 21] can be selected. These two are hardware intrusion detection systems (IDS) based on fixed subsets of Snort [43] rules, where the whole pattern (rule) matching process is realized directly in FPGA firmware.

Apart from the two abovementioned processing acceleration models, another approach commonly used in high-speed network monitoring is based on sampling of incoming data [20]. Here, the performance limitations of software applications are bypassed by processing of only a small fraction (a sample) of captured network traffic. Selection of sample to process can be totally random (e. g. every tenth packet) or conditioned by some easily obtainable packet features (e. g. a hash of selected header fields). A huge disadvantage of this approach lies in notably reduced precision or quality of obtained information because substantial chunks of data are just blindly dropped. For an application, this means uncontrolled loss of potentially interesting information that can lead to highly distorted measurements or even complete miss of a crucial event (e. g. network attack or anomaly).

All described models of hardware acceleration have significant shortcomings (in performance, flexibility or precision) preventing their effective utilization for 100 Gbps and faster networks. That is why a new acceleration approach needs to be designed – one that overcomes flexibility issues of fully hardware processing together with the inferior performance of purely software processing. Additionally, it should enable obtaining of high quality, unsampled, monitoring and security information.

## 2.5 Related Work

Apart from basic approaches to network monitoring and security in high-speed networks presented in the previous section, there are many sophisticated published works dealing with challenges of monitoring and security of high-speed networks. Therefore, several of these approaches that may to some extent resemble research area and objectives of this thesis are discussed. However, it is shown that those works have significant differences and shortcomings compared to this thesis.

Snort [43] is an open source software based network intrusion prevention and detection system. It relies only on regular expression matching, while this thesis does not want to enforce nor assume any particular type of software processing. Many papers dealing with hardware acceleration of Snort have been published, but they typically also restrict their focus to regular expression matching only. However, the area of network security monitoring is much richer than just that and this kind of limitation to only one specific kind of problems makes those systems insufficient for robust practical deployment. Similarly to Snort, L7-filter [10] also relies entirely on regular expressions. It is a Linux based packet classification software aiming at application layer protocol identification. A software library for application layer traffic processing called nDPI [34] provides an excellent example showing that regular expression matching alone is not sufficient. While this open source library is probably too complex to be fully hardware accelerated, it can certainly take advantage of some form of hardware acceleration provided by a kind of concept researched in this thesis.

The OpenSketch architecture [51] employs a configurable pipeline of hashing, classification and counting stages. These stages can be set to perform the computation of various network traffic statistics. OpenSketch is tailored to compute what its authors call sketches – probabilistic structures allowing measurement (estimation) and detection of various aspects of the network communication with a defined error rate. It is certainly not intended for complete flow-based monitoring, nor for precise and exact, error-free measurements. Also, OpenSketch does not define means to allow for application level protocol parsing nor other more sophisticated types of processing.

FlowContext system [22] provides a flexible way to implement stateful network traffic processing in an FPGA. Flow-based (specifically NetFlow) monitoring is among the examples of its use. However, it does not provide sufficient means for acceleration of general software applications. Therefore, FlowContext is not flexible enough to be effectively used for tasks exceeding the capabilities of a single FPGA.

There have been several efforts to accelerate NetFlow traffic monitoring in FPGAs, recently even as an open source project [16] for the NetFPGA platform [30]. This work is however aimed only at standard NetFlow and does not describe means for any kind of extended software processing nor further acceleration possibilities. On the other hand, the aim of this thesis is more towards flexible acceleration of wider range of application specific software processing.

The Shunt system [47] is a hardware accelerator that supports diversion of suspicious/interesting traffic to software for further analysis. To this end it partially resembles research objective of this thesis, however, Shunt accelerates only packet forwarding/filtering and does not include any possibilities of processing acceleration. Therefore, it is not very useful for the majority of network monitoring applications as they need some information about every packet.

Xilinx has recently announced SDNet environment [49] for software defined, hardware accelerated networking. The system relies upon high-level language to describe a network application, which is then compiled to a form of a hardware accelerator for a Xilinx FPGA. From the limited information, available at the time of writing, it seems that SDNet is not aiming at the definition of specific acceleration concept like this thesis, but rather can be used to describe custom hardware modules a bit easier. Another promising activity in this area is the definition of open P4 language standard [2], which is a high-level language specifically designed for the description of network traffic processing and forwarding. It enables protocol, vendor and target independent definitions thus can be also used for more comfortable programming of custom processing modules. But similar to SDNet, P4 does not define any particular acceleration architecture concept.

FlowSense [50] is a lightweight system aiming at estimating the network performance such as link utilization. It uses the built-in counters of OpenFlow switches to estimate the network parameters. While this approach brings virtually no overhead, its possibilities are limited by the OpenFlow protocol messages content and no other measurement can be done using this technique. There is no support for application level processing nor deeper packet inspection in FlowSense. Furthermore, it can be deployed only in SDN-based networks, whereas this thesis aims to more generally utilizable concept.

To summarize, none of the known state of the art approaches possess all the qualities required by the posed objective of this thesis research. Therefore, there is a valid need for the design of a novel acceleration concept for application layer analysis with high throughput, flexibility of use and ease of deployment for high-speed networks that enables obtaining of rich, unsampled monitoring and security information.

# Chapter 3

# Research Summary

## 3.1 Core Ideas

The design of the novel hardware acceleration concept for network monitoring applications leads off from the current state of the art ideas presented in the previous chapter. Also, typical characteristics of network traffic and information about how network monitoring and security applications usually process captured packets are considered. The designed acceleration concept must offer sufficient speed-up of applications for their deployment in high-speed networks. Also, it must be simply applicable and flexible enough to be able to support different kinds of applications. All of these parameters can be achieved using appropriate hardware software co-design approach inspired by key SDN features, as SDN achieves something similar in the area of network traffic routing by coupling of powerful hardware data processing with flexible software control (intelligence). Therefore, the designed concept resulting from this thesis research can be viewed as a natural extension of SDN ideas into the area of network monitoring and security.

Following the main idea of SDN, the hardware part of the designed acceleration concept realizes various types of network data preprocessing (functional base), but it leaves the management of this preprocessing utilization together with more advanced functionalities to the software part (intelligence). The management of the hardware preprocessing is centralized by a software controller, which mediate it through simple API directly to the running applications. Similar to OpenFlow, the controller is able to manage the preprocessing of network traffic at a flow level, enabling applications to precisely control behavior of the preprocessing according to the situation on the network and current needs. Utilization of various types of hardware traffic preprocessing relieves the software processing, and therefore, increases total throughput of the system. Furthermore, leaving all management and advanced processing to the software leads to high flexibility of the system. As the considered coupling of hardware and software functionality is very tight, an appropriate platform to implement the designed concept can be a hardware accelerated network interface card with FPGA plugged into a powerful commodity computer.

The achievable efficiency of the described acceleration approach closely depends on identification and selection of the appropriate traffic preprocessing methods for the hardware. The selection must be based on general data needs characteristics of standard network monitoring and security applications. Basic (flow based) network monitoring requires to have some information about each packet. This information is usually extracted from headers of the packets and payloads data are not interesting. Hardware can, therefore, extract and aggregate interesting information from packets and send only these data into the software in

unified form (like in [44] or [13]) instead of forwarding whole packets. Transfers of only selected information results in reduced PCI Express throughput requirements and host CPU load. On the other hand, network security and advanced monitoring applications (e. g. application layer analysis) usually perform a deeper inspection (DPI) of interesting packets that requires packet payload. To satisfy the needs of such applications, whole packets of the selected (potentially interesting) flows must be sent to the software, but the rest can be dropped. For example, detection of attacks on SSH service or monitoring of information in HTTP headers needs only a few initial packets of each flow.

Granular (per flow) utilization management of the described forms of hardware preprocessing enables for an interestingness based division of network traffic processing. Whole packets of the most interesting traffic groups are thoroughly analyzed in the software, less important packets are preprocessed and potentially aggregated in the hardware retaining only key parameters and uninteresting parts of traffic are directly dropped. This leads to controlled hardware accelerated reduction of network traffic via controlled loss of unimportant information. Also, it can be viewed as a form of informed data sampling that retains the full informational value of incoming traffic. Thus, software applications process smaller data volumes with higher information density and, therefore, the effectiveness of CPU utilization and PCI Express bandwidth allocation is raised. On the current multicore platforms, the CPU utilization effectiveness can be further increased by hardware division of software bound data into multiple independent communication (DMA) channels (similar to [44]). This division should be also managed on a per flow basis by the controller based on applications needs.

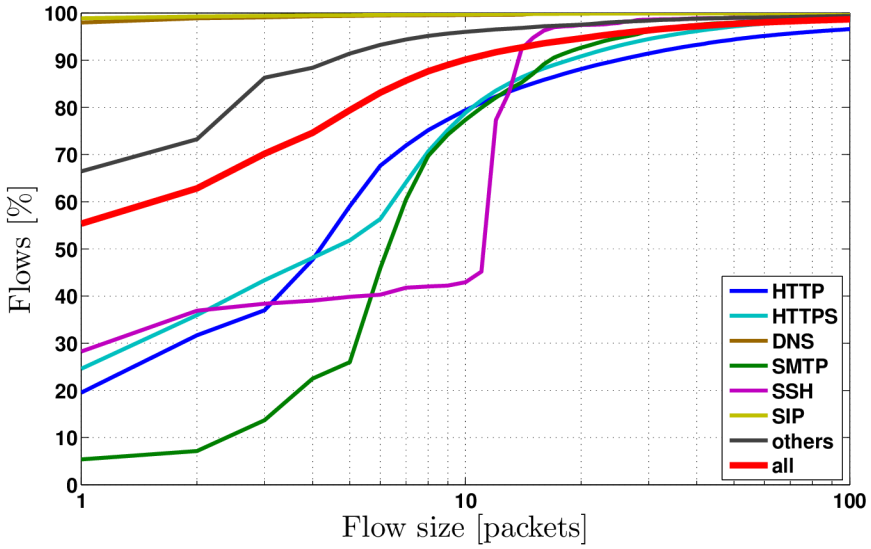


Figure 3.1: Cummulative distribution functions of flow sizes.

Another parameter that closely influences the achievable efficiency of the proposed acceleration approach are characteristics of high-speed network traffic. As software applications manage the preprocessing of packets at a flow level, benefits achievable from each offload rule are directly proportional to size/weight of (number of packets in) that flow. Basic information from flow sizes measurement can be seen in figure 3.1. Each line of the graph shows the percentage of flows that consists of fewer packets than a given number on the x-axis. On average (red thick line) only a tenth of all network flows have more than 10 packets. Also,

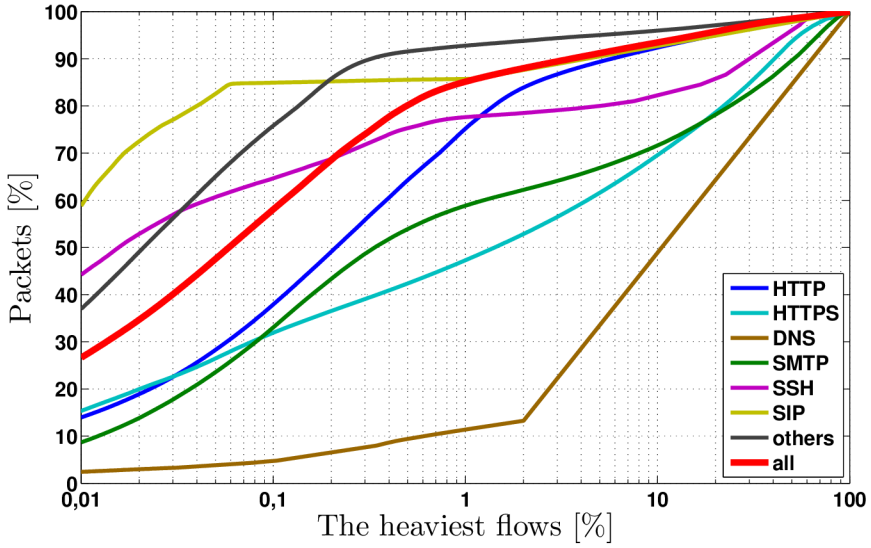


Figure 3.2: Portions of packets carried by the percentage of the heaviest flows.

virtually all flows of selected services (such as DNS and SIP) consist of a single packet. But, figure 3.1 does not clearly say anything about the percentage of all packets carried by flows of different sizes. It is known that high-speed network traffic has a heavy-tailed character of flow size distribution [15, 25]. The heavy-tailed character of flow size distribution derived from the measured values is shown in figure 3.2. The graph shows the portions of all packets carried by the specified percentage of the heaviest flows on the network. It can be seen that on average (red thick line) 0.1 % of the heaviest flows carry around 60 % of all packets and 1 % carry even around 85 %. An exception to the heavy-tailed distribution of flow sizes is the DNS service. This observed heavy-tailed character of network flows has the following potentially positive consequence for an achievable efficiency of researched approach: even if applications select only a small percentage of flows for offload, processing of majority of packets is still accelerated.

To summarize, the research presented in this thesis is centered around design, features, and feasibility of a novel acceleration concept based on key ideas presented in this section so far. Therefore, a concept of hardware accelerated flow based division and informed reduction of network traffic manageable on-the-fly according to needs of various software based monitoring and security applications without the reduction in their accuracy. This researched concept has been labeled as *Software Defined Monitoring – SDM*.

## 3.2 Research Process

Initial research process that gradually transformed the SDM concept from the already described set of functional ideas into a fully specified design of acceleration platform concept is in steps described by publications [p2, p5, Paper1]. The most complete and detailed description of main SDM design is provided in [Paper1], making it the cornerstone of this thesis research. Apart from SDM description, the paper also elaborates the concept feasibility by identifying its potential weak spots and their effects on the achievable performance based on detailed analysis of various real network traffic characteristics. Furthermore, some preliminary performance measurements are presented.

To further extend the deployment flexibility and acceleration potential of the designed SDM, possibilities of High-Level Synthesis (HLS) utilization to enhance capabilities of hardware are evaluated in [Paper2]. Using this approach, SDM concept can be easily extended to accelerate even applications with specific (uncommon) network traffic preprocessing requirements. An example of such extension creation is described in [Paper3], where specialization of SDM system for change-point anomaly detection method NP-CUSUM is described.

Another researched area of SDM improvements concerns approaches used in subcomponents of its FPGA architecture. Generally utilizable subcomponents were preferred here, as their optimizations do not aid only SDM implementation itself, but a broader spectrum of accelerated networking designs can be enhanced by them. More precisely, novel architectures for analysis (parsing) of packet headers [p1, Paper4] and packet classification (filtering) [Paper5] were designed and created. Unique modular FPGA architecture of packet headers parser is proposed in [p1]. A more detailed description, with some additional enhancements, is provided in [Paper4]. The key idea of designed parser's functionality is also patented by American patent no. US 8,923,300 B2. Packet classification approach designed for needs of SDM is based on cuckoo hashing principle [38]. Description of unique FPGA architecture implementing the cuckoo hashing principle including its reconfiguration procedures is part of [Paper5]. As various kinds of hash tables are commonly used in many classification and filtering engines, effective hash functions realizations for FPGAs were also explored. CRC functions can be applicable here, because their implementations in FPGA can utilize, in networking architectures usually unused, DSP blocks as shown by [p3, p4].

A bit specific area of research conducted on behalf of SDM creation and assessment revolved around the need for an FPGA card for 100 Gbps Ethernet, which was nonexistent at the time. As part of my involvement in research of Cesnet association, I helped in the creation of such a card and also demonstrated its working prototype to the scientific community at FPL 2014 conference in Munich (together with already mentioned [Paper2]). Some details about the card can be found in the paper [p6]. Mentioned Cesnet's card was the first working FPGA accelerated network interface card in the world, that was capable of 100 Gbps Ethernet wire-speed packet capture. Furthermore, a prototype implementation of SDM concept, created to enable real network experiments, utilizes this FPGA card. Details about and showcasing of the prototype make up the content of publication [p7].

Finally, a more detailed summary of the whole research around SDM is composed into [Paper6], making it the pinnacle of this thesis. Apart from information from previous papers, it also proposes some additional SDM enhancement like auto-adaptive heavy network flows detection heuristic and provides results of additional real network experiments.

## 3.3 Papers

This section contains brief descriptions and abstracts of all included papers. The description of each paper starts with its motivation and ends with highlighted summary of its main contributions toward the research presented in this thesis. Full texts of all the included papers in their original formatting can be found in appendix A.

## Paper 1

### Software Defined Monitoring of Application Protocols

(Appendix section A.1) The task of network traffic monitoring is one of the key concepts in modern network engineering and security. But, currently used monitoring methods usually provide only a very limited view of the network. Therefore, the ability to improve the quality and flexibility of network monitoring by deeper packet inspection is very important. This creates an issue of obtaining the additional data without significant degradation of performance while operating with limited computational resources. Pure software and hardware approaches to this problem have each their respective shortcomings. That is why this paper proposes the idea of a hardware accelerator tightly coupled to a software controller with monitoring applications as software plugins (core SDM concept). The focus is the process of obtaining the high-quality, unsampled flow measurement data augmented by application layer information.
*The key contribution of this work is a design of the main SDM concept – a new extensible high-speed network monitoring concept, that includes a design of a new application-specific processor for the stateful flow measurement. Other results include assessment of the SDM system feasibility based on the analysis of network traffic characteristics and evaluation of the SDM prototype implementation in several use case scenarios.*


### Abstract

Current high-speed network monitoring systems focus more and more on the data from the application layers. Flow data is usually enriched by the information from HTTP, DNS and other protocols. The increasing speed of the network links, together with the time consuming application protocol parsing, require a new way of hardware acceleration. Therefore we propose a new concept of hardware acceleration for flexible flow-based application level monitoring which we call Software Defined Monitoring (SDM). The concept relies on smart monitoring tasks implemented in the software in conjunction with a configurable hardware accelerator. The hardware accelerator is an application-specific processor tailored to stateful flow processing. The monitoring tasks reside in the software and can easily control the level of detail retained by the hardware for each flow. This way the measurement of bulk/uninteresting traffic is offloaded to the hardware while the advanced monitoring over the interesting traffic is performed in the software. The proposed concept allows one to create flexible monitoring systems capable of deep packet inspection at high throughput. Our pilot implementation in FPGA is able to perform a 100 Gb/s flow traffic measurement augmented by a selected application-level protocol parsing.


## Paper 2

### Trade-offs and Progressive Adoption of FPGA Acceleration in Network Traffic Monitoring

(Appendix section A.2) The main motivation for the research described in this paper is to enhance application acceleration capabilities of SDM concept proposed by the previous paper. The aim is to strike a balance between the system throughput and its flexibility/programmability, to offer a configurable trade-off to the above, but mainly to endorse a progressive adoption of network monitoring subtasks to the hardware accelerator, driven solely by the needs of the networking community. With key requirements being the possi-

bility of rapid development and deployment of new monitoring applications and simplicity of their further speed-up by the relocation of specific processing routines directly into the hardware accelerator.
*The contributions of this work are SDM concept extensions. Firstly, by the software plugins that enable adjustment of SDM functionality as a reaction to future network threats. And more importantly, by the possibility of a custom preprocessing instructions definitions in the hardware accelerator, which can redefine the nature of the acceleration itself and can be easily implemented using HLS.*


### Abstract

Current hardware acceleration cores for network traffic processing are often well optimized for one particular task and therefore provide high level of hardware acceleration. But for many applications, such as network traffic monitoring and security, it is also necessary to achieve rapid development cycle to provide fast response to security threats. While high level synthesis tools allow to directly generate hardware architecture from C description, the required level of hardware expertise is still above that of a typical network security expert. Therefore we propose and evaluate a new concept of hardware acceleration for flexible flow-based network traffic monitoring with support of application protocol analysis. We leverage the existing hardware architectures for packet header parsing, classification etc. to create a complete system which allows easy dealing with the whole family of problems in the network traffic monitoring. The concept is called Software Defined Monitoring (SDM) and it relies on a configurable hardware accelerator implemented in FPGA, coupled with smart monitoring tasks running as software on general CPU. The monitoring tasks in the software control the level of detail and type of information retained during the hardware processing. This arrangement allows rapid application prototyping in the software, followed by possible further shifting of the timing critical parts of the processing to the hardware accelerator. The concept is proposed with the scalability in mind, therefore it is suitable for different FPGA based platforms ranging from embedded single-chip solutions (such as Zynq or Cyclone V) to high-speed backbone network monitoring boxes. Our pilot high-speed implementation using FPGA acceleration board in a commodity server is able to perform a 100 Gb/s flow traffic measurement augmented by a selected application-level protocol analysis.


## Paper 3

### FPGA Accelerated Change-Point Detection Method for 100 Gb/s Networks

(Appendix section A.3) As computer networks are getting faster, there is a need to analyze larger volumes of data for detection of network attacks and traffic anomalies. Therefore, to realize real-time detection of attacks on high-speed computer networks, the detection methods must be directly deployed in and accelerated by hardware monitoring probe. SDM concept seems as a promising architecture for such monitoring probe, as it enables detection methods to achieve analysis of unsampled high-speed network traffic without packet loss thanks to the possibility of processing routines relocation directly into SDM hardware.
*The main contribution of this work is hardware accelerated realization of anomaly detection method (NP-CUSUM) for high-speed networks as hardware plugin for SDM. The key result for this thesis is the provision of experimental results supporting the feasibility of HLS and*

*SDM combination for effective acceleration of even very specific traffic analysis and anomaly detection methods.*

**Abstract**

The aim of this paper is a hardware realization of a statistical anomaly detection method as a part of high-speed monitoring probe for computer networks. The sequential Non-Parametric Cumulative Sum (NP-CUSUM) procedure is the detection method of our choice and we use an FPGA based accelerator card as the target platform. For rapid detection algorithm development, a high-level synthesis (HLS) approach is applied. Furthermore, we combine HLS with the usage of Software Defined Monitoring (SDM) framework on the monitoring probe, which enables easy deployment of various hardware-accelerated monitoring applications into high-speed networks. Our implementation of NP-CUSUM algorithm serves as hardware plugin for SDM and realizes the detection of network attacks and anomalies directly in FPGA. Additionally, the parallel nature of the FPGA technology allows us to realize multiple different detections simultaneously without any losses in throughput. Our experimental results show the feasibility of HLS and SDM combination for effective realization of traffic analysis and anomaly detection in networks with speeds up to 100 Gb/s.

## Paper 4

### Design Methodology of Configurable High Performance Packet Parser for FPGA

(Appendix section A.4) Since computer networks evolve both in terms of speed and complexity, there is still a need for packet parsing modules at all points of the infrastructure. Also, there are very different expectations on packet parsers in terms of latency, throughput and area tradeoffs. With the recent rise of SDN, new protocols appear at an even faster rate as before. This trend favors flexible parser architectures with a configurable set of supported protocols, what is often solved only partially.
*The main contribution of this work is the introduction of a highly configurable modular packet parser design for FPGAs, which achieves throughputs sufficient for high-speed networks (hundreds of Gbps) and is usable in a wide variety of FPGA architectures including SDM accelerator.*

**Abstract**

Packet parsing is among basic operations that are performed at all points of a network infrastructure. Modern networks impose challenging requirements on the performance and configurability of packet parsing modules. However, high-speed parsers often use a significant amount of hardware resources. We propose a novel architecture of a pipelined packet parser for FPGA, which offers low latency in addition to high throughput (over 100 Gb/s). Moreover, the latency, throughput and chip area can be finely tuned to fit the needs of a particular application. The parser is hand-optimized thanks to a direct implementation in VHDL, yet the structure is uniform and easily extensible for new protocols.

## Paper 5

### Fast Lookup for Dynamic Packet Filtering in FPGA

(Appendix section A.5) The speed and complexity of networks are growing rapidly, creating a demand for new approaches to a high-speed packet processing. One major trend is to keep hardware simple by offloading the complexity of a control path into the software (e.g. SDN or SDM). The offload requires the hardware to look up a piece of data (e.g. action, state) associated with a flow key (e.g. IP address or tuple of addresses, ports, and protocol) per each arriving packet. This is also essential for a wide variety of other existing applications (e.g. NATs, firewalls, probes). Various approaches to fast lookup have been proposed with commonly occurring drawbacks of poor memory utilization or slow lookup requiring multiple memory accesses. Also, the capability of rapid on-the-fly updates of active lookup rules is usually not addressed at all. This paper proposes a novel hardware lookup concept designed to solve the aforementioned issues.
*The main contribution of this work is the proposal of a fast lookup concept for FPGA-oriented platforms, that provides efficient utilization of memory and logic resources. The lookup concept has some unique features favorizing its usage in SDM – possible utilization of external memory to store even large sets of rules and FPGA implementation of rule reconfiguration logic enabling frequent on-the-fly updates of the active rule set.*


### Abstract

Rapidly growing speed and complexity of computer networks impose new requirements on fast lookup structures which are utilized in many networking applications (SDN, firewalls, NATs, etc.). We propose a novel lookup concept based on the well-known cuckoo hashing, which can achieve good memory utilization, supplemented by a binary search tree for offloading the colliding keys and supporting LPM lookup. We also propose a hardware architecture implementing this lookup concept in the FPGA. Our solution is suitable for lookup of the variable-length keys in 100+ Gbps networks. Memory utilization of the proposed concept is thoroughly evaluated and it is shown that the concept is scalable to external memory components.


## Paper 6

### Software Defined Monitoring of Application Protocols

(Appendix section A.6) As already mention, modern network engineering and security heavily rely on the network traffic monitoring. Therefore, monitoring information of the highest quality are required, but they can be obtained only by processing of unsampled network traffic. While many researchers focus on harvesting knowledge from statistical flow-based monitoring data, this paper argues that the ability to analyze application layer in the monitoring process is also crucial for the improvement of the quality and flexibility of network monitoring. Enrichment of monitoring data with added information from the application layer (or other sources) is crucial for better detections of various network threats (e.g. Heartbleed). Required deeper packet inspection realized with a commodity CPU has limited performance and using solely hardware architecture leads to flexibility issues. Therefore, a proposal of appropriate software hardware co-design concept is required. Furthermore, to achieve practically deployable system, the concept must not neglect flexibility, ease of use and speed of response to newly emerged problems.

*This work best summarizes the conducted research of this thesis in all its final depth. Main contributions include interconnection of all proposed features into a single more robust SDM concept description, detailed synthesis results about FPGA resources utilization, evaluation of additional network security use case scenarios, extended evaluation of SDM prototype deployment in real networks and better positioning of our research in the field thanks to extensive related work analysis.*

**Abstract**

With the ongoing shift of network services to the application layer also the monitoring systems focus more on the data from the application layer. The increasing speed of the network links, together with the increased complexity of application protocol processing, require a new way of hardware acceleration. We propose a new concept of hardware acceleration for flexible flow-based application level traffic monitoring which we call Software Defined Monitoring. Application layer processing is performed by monitoring tasks implemented in the software in conjunction with a configurable hardware accelerator. The accelerator is a high-speed application-specific processor tailored to stateful flow processing. The software monitoring tasks control the level of detail retained by the hardware for each flow in such a way that the usable information is always retained, while the remaining data is processed by simpler methods. Flexibility of the concept is provided by a plugin-based design of both hardware and software, which ensures adaptability in the evolving world of network monitoring. Our high-speed implementation using FPGA acceleration board in a commodity server is able to perform a 100 Gb/s flow traffic measurement augmented by a selected application-level protocol analysis.

## 3.4 List of Publications

### Papers Included in Thesis

[Paper1] Lukáš Kekely, Viktor Puš, and Jan Kořenek. Software defined monitoring of application protocols. In *Proceedings of IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, pages 1725–1733, Toronto, CA, USA, 2014. IEEE Computer Society. ISBN: 978-1-4799-3360-0.

[Paper2] Lukáš Kekely, Viktor Puš, Pavel Benáček, and Jan Kořenek. Trade-offs and progressive adoption of FPGA acceleration in network traffic monitoring. In *24th International Conference on Field Programmable Logic and Applications*, pages 1–4, Munich, Germany, 2014. IEEE. ISBN: 978-3-00-044645-0.

[Paper3] Tomáš Čejka, Lukáš Kekely, Pavel Benáček, Rudolf B. Blažek, and Hana Kubátová. FPGA accelerated change-point detection method for 100Gb/s networks. In *9th Doctoral Workshop on Mathematical and Engineering Methods in Computer Science*, pages 40–51, Brno, Czech Republic, 2014. NOVPRESS. ISBN: 978-80-214-5022-6.

[Paper4] Lukáš Kekely, Viktor Puš, and Jan Kořenek. Design methodology of configurable high performance packet parser for FPGA. In *17th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems*, pages 189–194, Warszaw, Poland, 2014. IEEE Computer Society. ISBN: 978-1-4799-4558-0.

[Paper5] Lukáš Kekely, Martin Žádník, Jiří Matoušek, and Jan Kořenek. Fast lookup for dynamic packet filtering in FPGA. In *17th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems*, pages 219–222, Warszaw, Poland, 2014. IEEE Computer Society. ISBN: 978-1-4799-4558-0.

[Paper6] Lukáš Kekely, Jan Kučera, Viktor Puš, Jan Kořenek, and Athanasios V. Vasilakos. Software defined monitoring of application protocols. *IEEE Transactions on Computers*, 65(2):615–626, 2016. ISSN: 0018-9340.

### Other Relevant Papers

[p1] Lukáš Kekely, Jan Kořenek, and Viktor Puš. Low-latency modular packet header parser for FPGA. In *Proceedings of the Eighth ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, pages 77–78, New York, NY, USA, 2012. ACM. ISBN: 978-1-4503-1685-9.

[p2] Lukáš Kekely and Viktor Puš. Software defined monitoring. In *TERENA Networking Conference 2013*, Maastricht, Netherlands, 2013. TERENA.

[p3] Viktor Puš, Lukáš Kekely, and Tomáš Závodník. Using DSP blocks to compute CRC hash in FPGA. In *The 2014 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, page 256, Monterey, CA, USA, 2014. ACM. ISBN: 978-1-4503-2671-1.

[p4] Tomáš Závodník, Lukáš Kekely, and Viktor Puš. CRC based hashing in FPGA using DSP blocks. In *17th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems*, pages 179–182, Warszaw, Poland, 2014. IEEE Computer Society. ISBN: 978-1-4799-4558-0.

[p5] Lukáš Kekely. Software defined monitoring: The new approach to high-speed network monitoring. In *Computer Architectures and Diagnostics*, pages 74–79, Liberec, Czech Republic, 2014. Technical University of Liberec. ISBN: 978-80-7494-027-9.

[p6] Viktor Puš, Petr Velan, Lukáš Kekely, Jan Kořenek, and Pavel Minařík. Hardware accelerated flow measurement of 100G ethernet. In *IFIP/IEEE International Symposium on Integrated Network Management (IM 2015)*, pages 1147–1148, Ottawa, Canada, 2015. IEEE Xplore Digital Library. ISBN: 978-3-901882-76-0.

[p7] Viktor Puš, Lukáš Kekely, Jan Kučera, and Denis Matoušek. Live demonstration of application layer traffic monitoring at 100 Gbps. In *Proceedings of the 40th Annual IEEE Conference on Local Computer Networks (LCN 2015)*, pages A50–A51, Leonia, NJ, USA, 2015. IEEE. ISBN: 978-1-4673-6769-1.

[p8] Lukáš Kekely, Viktor Puš, Jakub Cabal, and Jan Kořenek. Multi buses: Theory and practical considerations of bus width scaling in FPGAs. In *27th International Conference on Field Programmable Logic and Applications*, Ghent, Belgium, 2017. IEEE. (submitted).

# Chapter 4

# Discussion and Conclusions

This chapter provides a discussion of the research presented in this thesis. It also summarizes the achieved results, draws their conclusions and finally, presents possible directions for future research.

This thesis is focused around design and evaluation of a novel concept of Software Defined Monitoring – a software controlled (defined) hardware accelerated network traffic processing for network monitoring and security applications. First, a study of current trends in network monitoring, security and software defined networking, together with, an analysis of existing scientific or commercially available traffic processing acceleration techniques has been performed. Obtained information are summarized at the beginning of this thesis (chapter 2) and have been used as an inspiration in the design of the proposed approach. Another valuable information for the design process has been provided by performed study of network traffic's main characteristics. In order to obtain these characteristics and also for later evaluation of the proposed acceleration concept, a direct access to real high-speed network data must have been obtained. Therefore, a collaboration with Cesnet association was established and all of the measurements presented in this thesis were conducted in high-speed CESNET2 backbone network. CESNET2 is Czech National Research and Educational Network (NREN) which has optical links operating at speeds up to 100 Gbps and routes mainly IP traffic. It connects 27 institutions to the Internet and serves around 200,000 users. Finally, all specific research steps conducted to design, optimize and extend the proposed SDM acceleration concept are presented in descriptions of selected published papers provided in the previous chapter of this thesis.

## 4.1 Results

In order to evaluate the proposed SDM concept, its prototype has been implemented. The hardware part of the prototype is realized on an accelerator network interface card from COMBO family with a powerful Virtex-7 H580T FPGA. The FPGA firmware realizes the main SDM acceleration functionality, such as packet header parsing and NetFlow statistics aggregation, but also 100 Gbps Ethernet packet capture, PCI-Express and QDR external memory interface controllers. The software is realized as an extensible flow exporter with application specific behaviors realized as a set of plugins. This arrangement allows modification of its functionality to the extent required by the SDM concept.

### Acceleration Evaluation

The SDM concept has been evaluated in various realistic use case scenarios in order to measure achievable acceleration. These include the following cases:

- **Basic NetFlow measurement.** All packets from a network line must be taken into account. Packet headers only are sent to the software by default and the software applications adds dynamic rules to offload whole NetFlow measurement of selected flow into the hardware accelerator.
- **Port scan detection.** This use case demonstrates a measurement that is flow-based, yet not directly NetFlow-like. The software application observes headers of the first several packets of each flow and installs drop rules for the subsequent packets of selected flows. This information is typically enough to detect port scan attacks through various methods.
- **Heartbleed detection** clearly demonstrates the need for application layer processing in the network security monitoring. The software application first instructs the accelerator to drop all non-SSL packets. Then further rules to drop packets of heavy SSL flows are installed in the runtime because the Heartbleed attack can be detected by observing the first few packets of each flow.
- **HTTP header analysis.** Another application layer protocol example – parsing of HTTP headers and extraction of some interesting information (e.g. URL, host, user-agent). HTTP traffic is dominant in current networks, therefore, acceleration of its analysis is of high importance. Only the packets with a source or destination port 80 are sent to the software by default, others are dropped in the hardware. Furthermore, the application adds dynamic rules to drop the packets of HTTP flows in which it already detected and parsed the HTTP header.
- **NetFlow measurement enriched by HTTP analysis.** This case combines two of the previous use cases. Both NetFlow exporter and HTTP parser are active at the same time and their traffic preprocessing requirements are automatically combined by the SDM controller.

Tables 4.1 shows the results of the SDM system testing in the described use cases. The table shows portions of all captured packets preprocessed in the hardware by each preprocessing method. These utilizations of hardware preprocessing lead to a reduction of software application load displayed in table 4.2. The table shows portions of incoming packets and bytes that are processed by software applications in each use case relative to the state without the SDM accelerator. It also shows a percentage of flows for which a rule was created in the hardware.

Basic NetFlow measurement is significantly accelerated by the hardware flow cache. This way, the software application load is reduced to only a fifth of all packets. Further acceleration rises from the fact that only header and flow records are sent to the software, instead of complete packets. Similarly, in the Port scan scenario headers are sufficient and the unnecessary packets are dynamically dropped (not even aggregated). Therefore, the software does not parse packets anymore in both cases and the PCI Express bus load is reduced to less than one percent.

Dropping the packets based on static and dynamic rules is the preferred method of acceleration in both application layer parsing scenarios – Heartbleed detection, and HTTP

| Use case | Preprocessing method [% of packets] | | | |
|---|---|---|---|---|
| | **None** | **Header** | **NetFlow** | **Drop** |
| NetFlow | – | 20.55 | 79.45 | – |
| Port scan | – | 17.54 | – | 82.46 |
| Heartbleed | 4.91 | – | – | 95.09 |
| HTTP | 22.82 | – | – | 77.18 |
| HTTP+NetFlow | 23.34 | 10.56 | 66.10 | – |

Table 4.1: Usage of hardware preprocessing.

| Use case | SW load [%] | | Flows covered by rules [%] |
|---|---|---|---|
| | **None** | **Bytes** | |
| NetFlow | 20.66 | 0.98 | 6.37 |
| Port scan | 17.54 | 0.86 | 6.53 |
| Heartbleed | 4.91 | 3.77 | 0.95 |
| HTTP | 22.82 | 27.82 | 1.98 |
| HTTP+NetFlow | 34.02 | 29.00 | 6.04 |

Table 4.2: Software load using SDM, relative to the state without the SDM acceleration.

analysis. This leads to the HTTP parser load being reduced to only about a quarter of all packets and bytes and even more significant reduction in the Heartbleed detection. When standard NetFlow measurement is added to application protocol parsing, the load of the software slightly rises.

Graphs in figure 4.1 show results of SDM prototype testing in the NetFlow use case in more details. In the graphs, we can see courses of packets preprocessing ability of SDM system during an entire day of NetFlow measurement. The majority of all received packets (black line) are processed in the firmware flow cache (red line), leaving only a small portion for software processing (blue line). Offloaded percentage of packets is always in the range from 70 to 85 % of total traffic and is shown in gray shade bar at the bottom of the grid.

In figure 4.2 the trade-off in CPU load is examined, since the management of rules in SDM controller represents an additional load on the CPU. The effect of SDM acceleration on CPU utilization savings is shown in the most difficult of tested use cases – NetFlow measurement coupled with HTTP analysis. The left half of the graph shows measured CPU load with enabled SDM (after initial stabilization), right half shows CPU load after SDM disabling (i. e. all processing is done on CPU). According to table 4.2, the software load in this use case is up to one-third of received packets and bytes when using SDM. This perfectly corresponds to the observed increase in CPU load for packet processing (red line) from 20 % to 60 % after SDM disabling. However, the SDM controller brings some additional overhead (blue line) from configuration of the hardware accelerator and aggregation of the applications requests (rules). In the end, total CPU load is 2-times lower when using SDM (black line). This graph also suggests that SDM is best suited for highly advanced software tasks which consume significant amount of CPU resources.

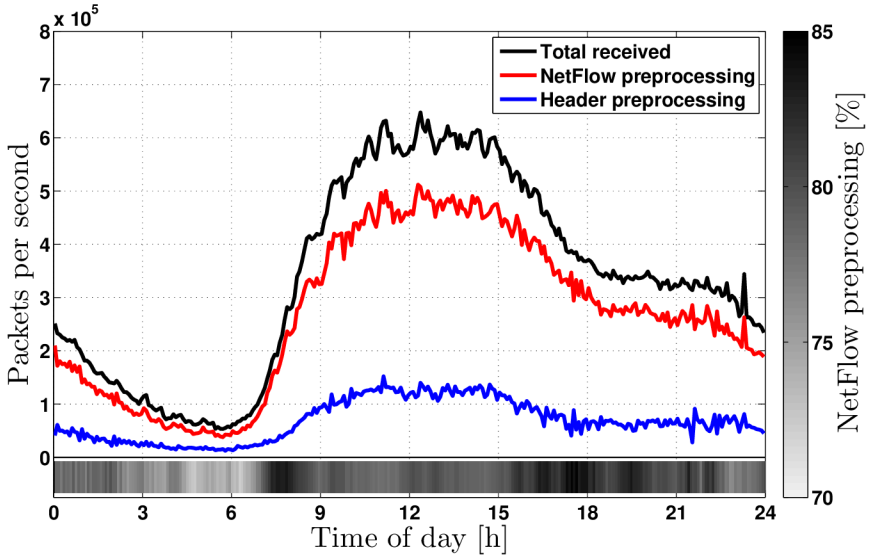


Figure 4.1: 24 hours NetFlow measurement with SDM.

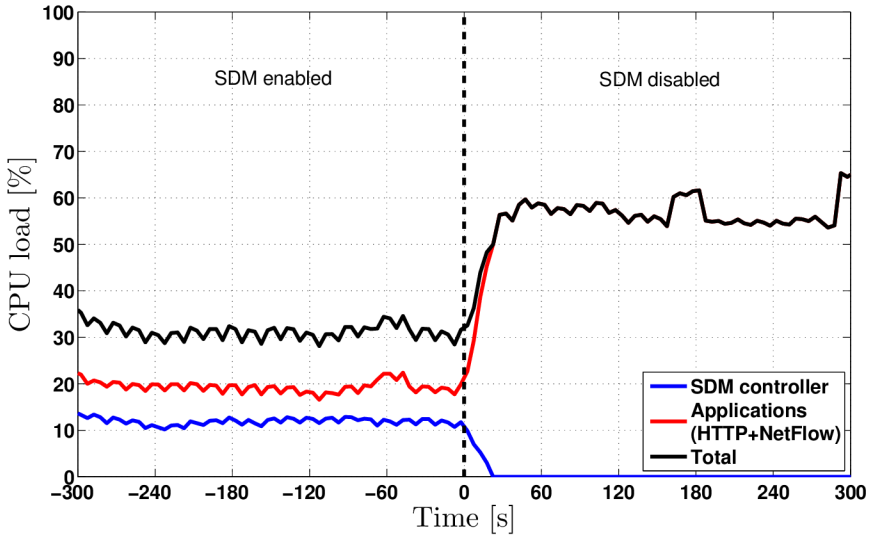


Figure 4.2: CPU load in HTTP and NetFlow use case with and without SDM support.

### FPGA Resources

Implemented high-speed SDM firmware runs at 200 MHz and occupies less than half of the available Virtex-7 H580T resources. A closer look at the FPGA resources requirements is presented in table 4.3. Using the same SDM core with a data width of 512 bits and throughput of 100 Gbps, 3 different FPGA architectures for cards with 3 different arrangements of Ethernet ports have been created: one 100 GbE port, two 40 GbE ports and eight 10 GbE ports. In addition to the high-performance 100 Gbps architecture, an analysis of the SDM core with narrower data width is also provided. These versions can be used in applications with lower throughput requirements, e.g. in embedded 1 or 10 Gbps probes. Note that the results for data widths other than 512 bits were obtained by simple downscaling of the SDM core and further optimizations are certainly possible in these cases.

| Firmware/Module | | Regs | LUTs | Throughput |
|---|---|---|---|---|
| Complete SDM | | 197 758 | 249 214 | 1×100 Gbps |
| | | 134 172 | 178 984 | 2×40 Gbps |
| | | 184 084 | 222 745 | 8×10 Gbps |
| SDM core | 512 b | 30 497 | 51 333 | 100 Gbps |
| | 256 b | 25 866 | 42 793 | 50 Gbps |
| | 128 b | 23 534 | 39 006 | 25 Gbps |
| | 64 b | 22 384 | 37 233 | 12.5 Gbps |
| | 32 b | 21 908 | 36 803 | 6.25 Gbps |
| Virtex-7 H580T FPGA | | 725 600 | 362 800 | |

Table 4.3: Resources of the SDM firmware

| Instruction | Regs | LUTs |
|---|---|---|
| NetFlow (handmade VHDL) | 1754 | 325 |
| NetFlow | 1846 | 824 |
| NetFlow Extended | 2070 | 1113 |
| TCP Flag Counters | 0 | 1046 |
| Timestamp Diff | 5199 | 2556 |
| Change-Point Detection | 5296 | 3919 |

Table 4.4: Resources of the instruction blocks

The feasibility of the SDM acceleration extensibility by application specific preprocessing instructions implemented using HLS is demonstrated by table 4.4. It shows the resource utilization of several selected instructions:

- **NetFlow** instruction is used for standard NetFlow aggregation. Its execution increases flow packet and byte counters, updates flow end timestamp and computes logical OR of the observed TCP flags.
- **NetFlow Extended** instruction has the same basic functionality as NetFlow. In addition, it stores TCP flags of the first five packets, what may become useful for analysis of TCP handshake or for detection of DoS attacks.
- **TCP Flag Counters** instruction performs increment of counters of individual observed TCP flags. Information from this aggregate can be used for various advanced flow analyses.
- **Timestamp Diff** instruction maintains records of inter-arrival times of the first eleven packets of the flow. These times can be used as network discriminators for flow-based classification or for identification of application protocol.
- **CPD** instruction (Change-Point Detection) shows implementation of more complex operation. CPD is a specific algorithm designed to detect network anomalies.

Extra resources required by these additional instructions are relatively small, compared to the whole firmware. Furthermore, a comparison between high-level synthesis and handmade implementation can be seen from the first two rows of the table. Handmade implementation occupies less than a half of LUTs and a bit fewer registers compared to HLS result. On the other hand, the creation of C implementation of the instruction and its subsequent automatic synthesis to HDL is much faster and simpler than HDL implementation.

## 4.2 Conclusions

From the results presented in the previous section, it can be concluded that the thesis confirmed the initial research hypothesis formulated in 1.1. Utilization of the designed SDM acceleration concept, indeed, shows a considerable increase in performance and information quality of various monitoring and security applications. Also, all defined research sub-goals has been successfully completed.

The main contribution of this thesis is the design of a brand-new concept of software controlled hardware acceleration of various monitoring and security applications called Software Defined Monitoring (SDM) that:

1. considerably increase achievable total performance or obtained level of details;
2. is flexible enough to aid wide range of tasks and applications;
3. can be easily utilizable without the need for application specific hardware changes, but also provides means for their usage to further increase performance;
4. is deployable in the fastest of the current high-speed networks.

Apart from the main contribution, a few additional general contributions can be derived from the proposed SDM concept and related research. They include:

- Detailed analysis of various characteristics of real high-speed network traffic.
- Design of a brand-new resource effective FPGA parser of packet headers with unique easily extensible modular architecture useful not only in SDM.
- Design of a novel high-throughput FPGA packet classification (filtering) architecture of the cuckoo hashing principle including unique on-chip realizations of its rule reconfiguration procedures.
- Exploration of effective CRC (XOR trees) based hash functions implementation possibilities using FPGA DSP blocks.

## 4.3 Deployment and Usage

As already mentioned, SDM concept has been tested in cooperation with Cesnet association using data from their backbone network. Promising acceleration results achieved by implemented SDM prototype (section 4.1) have sparked Cesnet's interest in potential deployment of the system. Cesnet's activities include protection of their network perimeter based on flow monitoring of traffic using accelerated probes on all peripheral high-speed links of their network. SDM can help to enhance the level of details retained in collected flow records by enabling execution of additional packet analyses on those probes (e. g. application layer processing or deep packet inspection). The process of deployment of SDM into Cesnet's productional network monitoring infrastructure is currently ongoing.

Through technology transfer agreement, Netcope Technologies have also shown interest in commercial applications of SDM. Netcope Technologies specializes in providing hardware accelerated high-speed and low latency network solutions. Their portfolio includes various network monitoring as well as hardware accelerated products. SDM is used as a basis for their commercially available Netcope Session Filter (NSF) [32]. NSF is a session-oriented

packet capture solution that uses hardware acceleration of per-packet processing and flow-based stateful filtering. It enables software applications to leverage hardware preprocessing of network flows and to identify flows of interest for further software processing.

Finally, various research activities and projects take advantage of SDM concept. In research project *Modern Tools for Detection and Mitigation of Cyber Criminality on the New Generation Internet* (VG20102015022) by Ministry of the Interior of the Czech Republic, first brief exploration of SDM utilization potential for lawful interception (LI) probes was performed and proven beneficial. Subsequently, research project *Smart Application Aware Embedded Probes* (VI20152019001) by Ministry of the Interior of the Czech Republic has been started. Its main goal is to create advanced embedded LI probes for 1 and 10 Gbps networks based on SDM acceleration of application layer processing.

## 4.4 Future Work

As speeds of computer networks are constantly raising, there is always a need for further speed-ups of SDM firmware. Standard of 400 Gbps Ethernet is expected within a year, with 1 Tbps Ethernet following shortly after. To accommodate such speeds, new interesting challenges arise in FPGA designs. One of the main being the widening of internal data path buses to such extent that threat of serious performance degradation from aliasing and alignment of the shortest frames seems unavoidable. To tackle this challenge, we propose a novel design method for the description of very wide buses which enable processing of multiple transactions on a bus per clock cycle (data word) [p8]. This method can be further refined and subsequently used to create SDM firmware (and its FPGA components) for even faster networks. Exploration of SDM potential in 400 Gbps networks should be a part of research project VI20172020064 by Ministry of the Interior of the Czech Republic.

An opposite approach can prove to be also interesting for further research – exploration of SDM usage potential in slower networks. This seems promising because various system on chip (SoC) architectures combining FPGA with CPU cores on a single chip are more and more common. Such SoCs are perfectly suited for creation of small embedded network devices based on SDM concept with SDM firmware and software parts tightly integrated into a single chip. Thanks to SDM acceleration, the performance of these embedded devices can prove to be sufficient for 1, 10 or even 40 Gbps networks. Thus, they would provide cheaper alternatives to conventionally used servers with network interface cards. Exploration of this research direction has already started as part of mentioned research project VI20152019001 by Ministry of the Interior of the Czech Republic.

Apart from retargeting to different network speeds, further possibilities of SDM deployment flexibility can be also explored. For example, the feasibility of SDM utilization in additional new use case scenarios can be evaluated. One of the main promising areas is acceleration of more complex IDS like Snort or Suricata [36] as a whole, not only specific threat detection methods. Another interesting example is a transformation of the whole SDM firmware description into some higher-level language. This would enable even easier adjustments of SDM firmware for various application-specific accelerations, thus enabling further speed-ups in many cases. P4 language [2] is becoming more and more promising in this area, especially after direct FPGA firmware generation from P4 has been introduced in [1]. Exploration of P4 description feasibility for SDM firmware has already started as part of research project TH02010214 by Technology Agency of the Czech Republic.

# Bibliography


[1] Pavel Benáček. *Generation of High-Speed Network Device from High-Level Description*. PhD thesis, Faculty of Information Technology, Czech Technical University in Prague, Czech Republic, 2017. URL: https://www.fit.cvut.cz/sites/default/files/PhDThesis-Benacek.pdf.

[2] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. P4: Programming protocol-independent packet processors. *SIGCOMM Computer Communication Review*, 44(3):87–95, July 2014. ISSN: 0146-4833.

[3] J.D. Case, M. Fedor, M.L. Schoffstall, and J. Davin. Simple network management protocol (SNMP). Request for Comments (RFC) 1157, Internet Engineering Task Force, May 1990. URL: http://www.ietf.org/rfc/rfc1157.txt.

[4] Alisha Cecil. A summary of network traffic monitoring and analysis techniques. Supervisor: Prof. Raj Jain, CSE WU in St. Louis, St. Louis, USA, 2006. URL: http://www1.cse.wustl.edu/~jain/cse567-06/net_monitoring.htm.

[5] Cisco DocWiki. Internetworking technology handbook: Security technologies. Cisco, October 2012. URL: http://docwiki.cisco.com/wiki/Security_Technologies.

[6] Cisco Networking Academy. *CCNA Exploration Course Booklet: Network Fundamentals, Version 4.0*. Course Booklets. Cisco Press, Indianapolis, USA, September 2009. ISBN: 978-1-58713-243-8.

[7] B. Claise. Specification of the IP flow information export (IPFIX) protocol for the exchange of IP traffic flow information. Request for Comments (RFC) 5101, Internet Engineering Task Force, January 2008. URL: http://www.ietf.org/rfc/rfc5101.txt.

[8] B. Claise. Cisco systems NetFlow services export version 9. Request for Comments (RFC) 3954, Internet Engineering Task Force, October 2004. URL: http://www.ietf.org/rfc/rfc3954.txt.

[9] Benoit Claise and Ralf Wolter. *Network Management: Accounting and Performance Strategies*. Cisco Press, 2006. ISBN: 1-587-05198-2.

[10] ClearFoundation. l7-filter. Online, October 2013. URL: http://l7-filter.clearos.com/.

[11] Frederic J. Cooper, Chris Goggans, John K. Halvey, Larry Hughes, Lisa Morgan, Karanjit Siyan, William Stallings, and Peter Stephenson. *Implementing Internet Security*. New Riders Publishing, Indianapolis, USA, June 1995. ISBN: 1-562-05471-6.

[12] S. Deering and R. Hinden. Internet protocol, version 6 (IPv6) specification. Request for Comments (RFC) 2460, Internet Engineering Task Force, December 1998. URL: http://www.ietf.org/rfc/rfc2460.txt.

[13] Pavel Čeleda, Milan Kováčik, Tomáš Koníř, Vojtěch Krmíček, Petr Špringl, and Martin Žádník. FlowMon probe. Technical Report 31/2006, CESNET, Prague, December 2006.

[14] Endace Technology Limited. DAG packet capture cards for packet monitoring tools, packet sniffers. Online, 2016. URL: https://www.endace.com/endace-dag-high-speed-packet-capture-cards.html.

[15] Cristian Estan and George Varghese. New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. *ACM Transactions on Computer Systems*, 21(3):270–313, August 2003. ISSN: 0734-2071.

[16] M. Forconesi, G. Sutter, S. Lopez-Buedo, and J. Aracil. Accurate and flexible flow-based monitoring for high-speed networks. In *23rd International Conference on Field Programmable Logic and Applications*, pages 1–4. IEEE, September 2013. ISBN: 978-1-4799-0004-6.

[17] Francesco Fusco and Luca Deri. High speed network traffic analysis with commodity multi-core systems. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, pages 218–224, New York, NY, USA, 2010. ACM. ISBN: 978-1-4503-0483-2.

[18] IEEE 802.3 Ethernet Working Group. IEEE 802.3 industry connections ethernet bandwidth assessment. Technical report, IEEE, San Diego, CA, USA, 2012. URL: http://www.ieee802.org/3/ad_hoc/bwa/BWA_Report.pdf.

[19] IEEE 802.3 Ethernet Working Group. IEEE P802.3bs: 200 Gb/s and 400 Gb/s ethernet task force. IEEE, May 2016. URL: http://www.ieee802.org/3/bs/.

[20] Ryszard Erazm Jurga and Miłosz Marian Hulbój. Packet sampling for network monitoring. Technical Report CH-1211, CERN – HP Procurve openlab project, Geneva, Switzerland, December 2007.

[21] Toshihiro Katashita, Yoshinori Yamaguchi, Atusi Maeda, and Kenji Toda. FPGA-based intrusion detection system for 10 gigabit ethernet. *IEICE Transactions on Information and Systems*, E90-D(12):1923–1931, 2007. ISSN: 1745-1361.

[22] Martin Košek and Jan Kořenek. Flowcontext: Flexible platform for multigigabit stateful packet processing. In *International Conference on Field Programmable Logic and Applications*, pages 804–807. IEEE, 2007. ISBN: 978-1-4244-1059-0.

[23] Jan Kořenek. *Fast Regular Expression Matching Using FPGA*. PhD thesis, Faculty of Information Technology, Brno University of Technology, Czech Republic, 2010. URL: http://www.fit.vutbr.cz/study/DP/PD.php?id=162.

[24] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach Featuring the Internet*. Addison Wesley, Boston, USA, first edition, July 2000. ISBN: 0-201-47711-4.

[25] Kun-chan Lan and John Heidemann. A measurement study of correlations of internet flow characteristics. *Computer Networks*, 50(1):46–62, January 2006. ISSN: 1389-1286.

[26] Liberouter / Cesnet TMC Group. NetCOPE. Online, January 2014. URL: https://www.liberouter.org/technologies/netcope/.

[27] Liberouter / Cesnet TMC Group. Hanic. Online, July 2012. URL: https://www.liberouter.org/technologies/hanic/.

[28] Petr Matoušek. Síťové aplikace a správa sítí: Měření provozu na síti pomocí NetFlow. FIT VUT v Brně, Brno, 2012.

[29] Abhishek Mitra, Walid Najjar, and Laxmi Bhuyan. Compiling PCRE to FPGAfor accelerating SNORT IDS. In *Proceedings of the 3rd ACM/IEEE Symposium on Architecture for Networking and Communications Systems*, ANCS '07, pages 127–136, New York, NY, USA, 2007. ACM. ISBN: 978-1-59593-945-6.

[30] J. Naous, G. Gibb, S. Bolouki, and N. McKeown. NetFPGA: Reusable router architecture for experimental research. In *Proceedings of the ACM Workshop on Programmable Routers for Extensible Services of Tomorrow*, pages 1–7. ACM, 2008. ISBN: 978-1-60558-181-1.

[31] Napatech A/S. The Napatech NAC - designed to accelerate your PCAP solution. Online, 2016. URL: https://www.napatech.com/products/napatech-nac/.

[32] Netcope Technologies. NSF-100G2: Netcope technologies session filter. Product brief, Netcope Technologies, a. s., Brno, January 2017.

[33] Stephen Northcutt, Lenny Zeltser, Scott Winters, Karen Kent, and Ronald W. Ritchey. *Inside Network Perimeter Security*. Sams Publishing, Indianapolis, USA, second edition, March 2005. ISBN: 0-672-32737-6.

[34] ntop. nDPI. Online, 2014. URL: http://www.ntop.org/products/ndpi/.

[35] ONF Market Education Committee. Software-defined networking: The new norm for networks. ONF white paper, Open Networking Foundation, Palo Alto, California, USA, 2012. URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf.

[36] Open Information Security Foundation. Suricata: Open source IDS / IPS / NSM engine. Online, 2016. URL: https://suricata-ids.org/.

[37] Open Networking Foundation. OpenFlow switch specification: Version 1.3.0. ONF OpenFlow spec, ONF, Palo Alto, California, USA, June 2012.

[38] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. In *Algorithms – ESA 2001*, volume 2161 of *Lecture Notes in Computer Science*, pages 121–133, Aarhus, Denmark, 2001. Springer. ISBN: 3-540-42493-8.

[39] J. Postel. User datagram protocol. Request for Comments (RFC) 768, Internet Engineering Task Force, August 1980. URL: http://www.ietf.org/rfc/rfc768.txt.

[40] J. Postel. Internet protocol. Request for Comments (RFC) 791, Internet Engineering Task Force, September 1981. URL: http://www.ietf.org/rfc/rfc791.txt.

[41] J. Postel. Transmission control protocol. Request for Comments (RFC) 793, Internet Engineering Task Force, September 1981. URL: http://www.ietf.org/rfc/rfc793.txt.

[42] J. Quittek, T. Zseby, B. Claise, and S. Zander. Requirements for IP flow information export (IPFIX). Request for Comments (RFC) 3917, Internet Engineering Task Force, October 2004. URL: http://www.ietf.org/rfc/rfc3917.txt.

[43] Snort Team. Snort – network intrusion detection & prevention system. Online, Cisco and/or its affiliates, 2014. URL: http://www.snort.org/.

[44] The Liberouter Project Team. Hashing network interface card handbook. Version 3.0.3, September 2014. URL: http://www.liberouter.org/package_releases/hanic-current/hanic-combov2-handbook.html.

[45] Michiel Uithol and Vincent van Kooten. Network monitoring based on flow measurement techniques. SURFnet research on networking, University of Twente, Enschede, The Netherlands, 2005.

[46] Ladislav Varga. Flexible network flow measurement. Master's thesis, Faculty of Information Technology, Brno University of Technology, Czech Republic, 2010. URL: http://www.fit.vutbr.cz/study/DP/DP.php.en?id=9407.

[47] Nicholas Weaver, Vern Paxson, and Jose M. Gonzalez. The shunt: An FPGA-based accelerator for network intrusion prevention. In *Proceedings of the 15th international symposium on Field programmable gate arrays*, pages 199–206, New York, NY, USA, 2007. ACM. ISBN: 978-1-59593-600-4.

[48] Nigel Williams, Sebastian Zander, and Grenville Armitage. A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification. *SIGCOMM Computer Communication Review*, 36(5):5–16, October 2006. ISSN: 0146-4833.

[49] Xilinx Inc. SDNet Development Environment: Expanding Programmability from the Control to the Data Plane. Online, 2014. URL: https://www.xilinx.com/products/design-tools/software-zone/sdnet.html.

[50] Curtis Yu, Cristian Lumezanu, Yueping Zhang, Vishal Singh, Guofei Jiang, and Harsha V. Madhyastha. Flowsense: Monitoring network utilization with zero measurement cost. In *Proceedings of the 14th international conference on Passive and Active Measurement*, pages 31–41, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN: 978-3-642-36515-7.

[51] Minlan Yu, Lavanya Jose, and Rui Miao. Software defined traffic measurement with OpenSketch. In *Proceedings of the 10th USENIX conference on Networked Systems Design and Implementation*, pages 29–42. ACM, April 2013. ISBN: 978-1-931971-00-3.

[52] N. Zilberman, Y. Audzevich, G. A. Covington, and A. W. Moore. NetFPGA SUME: Toward 100 gbps as research commodity. *IEEE Micro*, 34(5):32–41, Sept 2014. ISSN: 0272-1732.

# Appendix A

# Included Papers

## A.1 Paper 1

**Software Defined Monitoring of Application Protocols**

# Software Defined Monitoring of Application Protocols

Lukáš Kekely, Viktor Puš
CESNET a. l. e.
Zikova 4, 160 00 Prague, Czech Republic
Email: kekely,pus@cesnet.cz

Jan Kořenek
IT4Innovations Centre of Excellence
Faculty of Information Technology
Brno University of Technology
Božetěchova 2, 612 66 Brno, Czech Republic
Email: korenek@fit.vutbr.cz

***Abstract*—Current high-speed network monitoring systems focus more and more on the data from the application layers. Flow data is usually enriched by the information from HTTP, DNS and other protocols. The increasing speed of the network links, together with the time consuming application protocol parsing, require a new way of hardware acceleration. Therefore we propose a new concept of hardware acceleration for flexible flow-based application level monitoring which we call Software Defined Monitoring (SDM). The concept relies on smart monitoring tasks implemented in the software in conjunction with a configurable hardware accelerator. The hardware accelerator is an application-specific processor tailored to stateful flow processing. The monitoring tasks reside in the software and can easily control the level of detail retained by the hardware for each flow. This way the measurement of bulk/uninteresting traffic is offloaded to the hardware while the advanced monitoring over the interesting traffic is performed in the software. The proposed concept allows one to create flexible monitoring systems capable of deep packet inspection at high throughput. Our pilot implementation in FPGA is able to perform a 100 Gb/s flow traffic measurement augmented by a selected application-level protocol parsing.**

## I. Introduction

The task of network traffic monitoring is one of the key concepts in modern network engineering and security. A golden standard in the network monitoring is the basic NetFlow measurement. In NetFlow, the monitoring device collects basic statistics about the IP flows and reports them to a central storage collector in the Cisco NetFlow v5 protocol. NetFlow measurement is a stateful process, because for each packet the flow state record is updated in the device (e.g. counters are incremented), and only the resulting numbers are exported. This also implies that some information is lost in the monitoring process and that the flow collector (where further data processing is usually done) has a limited view on the network. The ability to analyze the application layer in the monitoring process is, therefore, very important in order to improve the quality and flexibility of network monitoring.

The evolution of the NetFlow protocol led to the IPFIX protocol [1]. IPFIX allows for the extension of the exported flow record for any other additional information. While IPFIX solves the task of *transmitting* the additional data, there remains the issue of *obtaining* the additional data. This process inevitably requires additional computational resources.

Pure software implementation of the application level flow monitoring is certainly possible, yet its throughput is limited mainly by the performance of commodity processors. It should be noted that every new packet is inevitably a cache miss in the CPU. Pure hardware implementation, on the other hand, has poor flexibility because the complex protocol parsers are very hard to implement in Hardware Description Languages. Moreover, the evolving nature of network threats and security issues implies the need for a fast change of the monitoring process, which is much more difficult for the hardware. These thoughts lead us to the idea of a hardware accelerator tightly coupled to a software controller with monitoring applications as software plugins.

We focus on the process of obtaining the high-quality, unsampled flow measurement data augmented by application-layer information. Our key idea is that even the advanced application-layer processing usually needs to observe only some flows containing only a small fraction of traffic (such as DNS, with typically no more than 1 % of all packets), or even only a small amount of packets within each of these flows (such as HTTP, typically carrying the HTTP header in the first few packets after the TCP handshake).

We employ a hardware accelerator to perform the offload of the flow measurement for the bulk traffic that is not (or no longer) interesting to the application-layer processing tasks. Also, the hardware accelerator partially has the role of the basic NIC - network interface card. Therefore, it passes a small fraction of the packets intact to the monitoring software and performs flow measurement of the rest.

The use of measurement offload can be easily controlled on a per flow basis by the monitoring software and adjusted to its current needs. Offload control is realized through unified interface by dynamically specifying a set of rules. These rules are then installed into the hardware accelerator to determine interestingness of individual network flows for advanced software processing. Thanks to this unified control interface the proposed system is very flexible and can be used for a wide range of different network monitoring applications. The whole system is designed to be easily extensible by monitoring plugins at the software side. Each monitoring application (in the form of SDM plugin) has three conceptual interfaces: input packets, output measured values, and the control interface to express interest and disinterest in particular fractions of the network traffic. We demonstrate the SDM system on four different monitoring applications: NetFlow measurement, HTTP parsing, a combination of both and DNS protocol parsing.

978-1-4799-3360-0/14/$31.00 ©2014 IEEE

The contribution of our work is three-fold:

- Design of a new concept of extensible high speed network monitoring system. This includes a design of a new application-specific processor for the stateful flow measurement and its controller software. (Chapter II)
- Analysis of network traffic to show the possibilities for the hardware acceleration. Assessment of the system feasibility is based on the analysis. (Chapter III)
- Implementation and evaluation of the system in several use cases. (Chapter IV)

## II. System Design

A standard model of the flow measurement widely used in 10 Gbps networks relies on a hardware network card performing a packet capture, sometimes enhanced by a packet distribution among several CPU cores. The captured traffic is then sent over the host bus to the memory, where packets are processed by the CPU cores. This model cannot be applied to 100 Gbps networks due to two major performance bottlenecks. First, the throughput of today's PCI Express busses is insufficient. The second bottleneck lies in limited computational power which is insufficient for advanced monitoring tasks. We propose a new acceleration model which overcomes the above-mentioned bottlenecks by a well-defined hardware/software co-design. The main idea is to give the hardware the ability to handle basic traffic processing. Only a granular control of the HW and some more advanced tasks are left for the software.

The basic idea of acceleration by the SDM system is based on a finely controlled data loss and data distribution realized by hardware preprocessing of the network traffic. The preprocessing is fully controlled by the software applications. Therefore, the first few packets of a new flow are sent to the software, which decides which type of hardware preprocessing will be used for the following packets of the flow. There are two basic options for the hardware acceleration:

- It is possible to extract the interesting data from packets in the hardware and send them only to the software in a predefined format, which we call a Unified Header (UH). Then only a few bytes for each packet are transferred through the PCI Express bus and the CPU has a lower load too because the packet parsing is done in the hardware.
- Furthermore, packets can be aggregated to NetFlow records directly in the HW which brings even higher performance savings.

Some advanced monitoring applications perform deep packet inspection on interesting fragments of traffic and, therefore, have to analyze the whole packets. For example, extraction of information from HTTP headers needs several first packets for each HTTP flow. Therefore, the proposed system provides a control over the hardware packet preprocessing at the flow level granularity.

The top-level conceptual scheme of the proposed SDM system is shown in Fig. 1. Data paths are represented by black arrows and control paths by red arrows. The system is composed of two main parts (firmware and software) connected together through the PCI Express bus. The processing of all incoming packets starts with the header parsing and extraction of interesting metadata (Header Field Extractor - HFE block). Extracted metadata are then used to classify the packet based on a software defined set of rules (Classifier block). Each rule identifies one specific flow and defines a method of hardware preprocessing of its packets. More precisely, each rule specifies the type of packet preprocessing and the target software channel. Packets can be processed in a hardware flow cache, dropped, trimmed or sent to the software unchanged or in the form of a Unified Header (UH Generator block). Flow records in the hardware flow cache are periodically exported to the software. Sending the data to the software is realized by the direct memory accesses (DMA) over the PCI Express bus. There are multiple independent logical DMA channels with the corresponding DMA buffers in the host RAM to aid parallel processing by a multicore CPU.

The data can be stored in DMA buffers in the form of whole packets, Unified Headers or flow records. This data can be monitored by the set of user specific software applications such as the flow exporter which analyzes the received data and exports the flow records to the collector. User applications can read the data from the selected DMA channels and can also specify which types of traffic they want to inspect and which flows can be preprocessed in hardware. For example, an HTTP header parser needs to inspect every packet in the HTTP flow until it acquires the required information (e.g. the URL). Definitions of interesting and uninteresting bulk traffic from all applications are passed to the SDM controller. The SDM controller aggregates the definitions into rules and configures the firmware behavior in order to achieve the maximal possible reduction of the traffic resulting in maximal hardware acceleration.

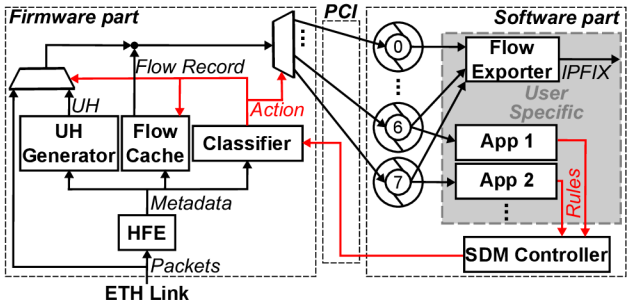


Fig. 1. Conceptual top-level scheme of SDM system

A different view of the proposed SDM system is shown as a layered scheme in Fig. 2. The SDM system is designed to work on a hardware accelerated network board with an FPGA chip. Our implementation uses a custom made board with 100 Gb/s Ethernet interface and Virtex-7 FPGA with the NetCOPE platform [2] realizing the basic network traffic capture and communication with the software (DMA). The core of the FPGA firmware is realized by the firmware part of the SDM system described earlier, which is able to process the incoming traffic at full speed of the network link. The software layer of the SDM includes means for the basic configuration of the firmware, network data transfer (black Data Path) and control of SDM firmware (red Control Path). Data can be received from the firmware in the standard PCAP or the proprietary SZE

format. On the top of the SDM system, there are individual user specific software applications.

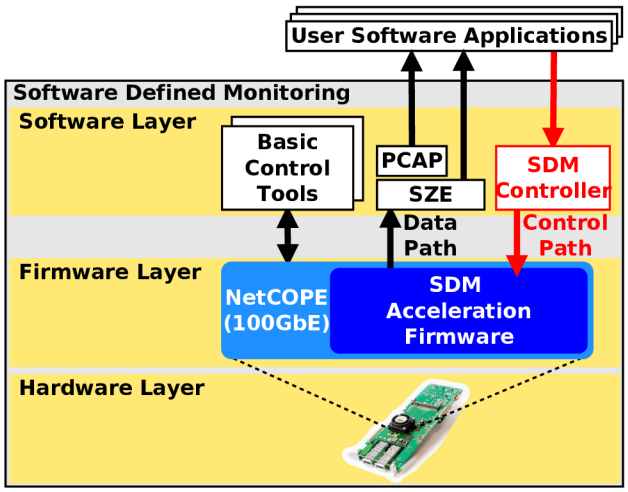


Fig. 2. Layered model of SDM acceleration system

Fig. 3 shows a top level implementation scheme of the SDM firmware. The main firmware functionality is realized by the processing pipeline of four modules: Header Field Extractor (HFE), Search, Update and Export. This pipeline processes the incoming network traffic and creates an outgoing data flow for the software. Incoming frames do not flow directly through the processing pipeline, but are rather stored in a parallel FIFO. The processing pipeline uses only metainformation extracted from frames headers (UH). Whole software control of the processing pipeline is managed by the SW Access module which configures preprocessing rules used in the Search unit. In order to achieve sufficient capacity for rules and flow records, the firmware stores them in external memory (Table1 and Table2). Access to the external memory is managed by Memory Arbiter.

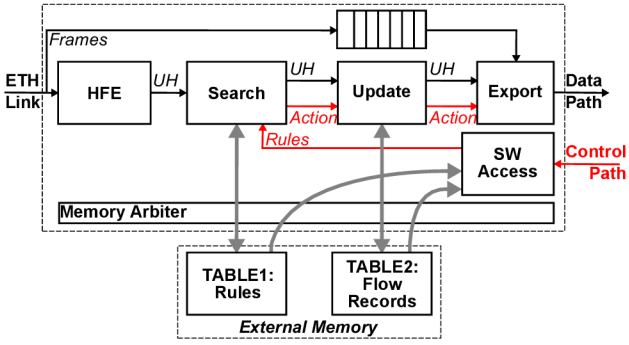


Fig. 3. Detailed firmware scheme

As already described, the SDM firmware functionality is realized by 6 main modules:

- **Header Field Extractor** analyzes headers of incoming frames and extracts interesting information from them, especially fields that clearly identify network flows. In order to identify flows we use the classical 5-tuple: source and destination IP addresses, source and destination TCP/UDP port numbers and a protocol number. We use our own flexible low latency modular implementation of the header parser [3].
- **Search** assigns an action to every processed frame based on its flow identifier. An action assignment is realized using a set of software defined rules in the form of a flow identifier paired with action (Table1 in external memory). Management of the rule set is possible through a control interface capable of an atomic add, remove or update of the rules. A frame classification by the Search unit works in 2 steps. Firstly, the frames are assigned with an action based on a small set of relatively static rules on flow groups (e.g. flows with source port 80). Secondly, the action from the first step can be further particularized by a set of dynamic rules for individual flows. Standardly, user applications set up rules of the first type during startup and then they manage the set of second type rules during traffic processing.
- **Update** manages the records for flows in Table2. It mainly actualizes their values based on input UH and its action. The action for every UH has the address of the record and a specification of the operation (aggregation type). Update of the record is realized by two memory operations: read actual values of the record fields and write back the updated values. Another operation is the export of the record values, possibly followed by the reset of the record values in the memory. Records can be exported not only at the flow end but also in a 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 SDM control software.
  In the first version of SDM we implement only the simple NetFlow aggregation as the record update operation – increase packets/bytes counters, update flow start/end timestamp and logical or of TCP flags. It is however possible to support more types of records and operations in the future.
- **Export** pairs together corresponding UH transaction with frame data from FIFO memory. Then it chooses the DMA channel and format for the data based on action assigned by the Search module.
- **SW Access** is the main access point into the SDM firmware from the software. Its primary function is to manage the rules and to initiate the export of the flow records based on software commands. Besides, it contains all state and control registers. It also enables direct software access into external memory (still used only for debug).
- **Memory Arbiter** provides and manages access to the external memory. Its main responsibilities are proper interleaving of memory accesses and routing of read data between units. It also ensures atomicity and deterministic succession of all memory operations.

The network traffic preprocessing by firmware is controlled from the software. The core of the controlling software are the

monitoring applications. Each monitoring application has the form of an SDM plugin. The main input to the plugin is the data path carrying the packets, UHs or flow records. The plugin output is the data that the plugin has parsed/detected/measured. This output data is then added to the exported IPFIX flow record. The third interface of the monitoring application is the interface to the SDM Controller.

From the application view, the SDM controller accepts the preprocessing requests from multiple applications, aggregates them and administers them into the firmware. In order to achieve that, the controller performs the following operations:

- On the fly management of the set of applications currently controlling the firmware preprocessing.
- Preprocessing requests reception from applications.
- Storing and aggregation of the received preprocessing rules (requests).
- Timed expiration of application rules.

The aggregation of preprocessing rules is based on different degrees of data reduction. Ordered from the lowest degree of data reduction the preprocessing types are: none (whole packets), partial (UH), complete (flow record) and elimination (packet drops). Therefore, aggregation of rules in the SDM controller is done simply by the selection of the lowest preprocessing degree (highest data preservation) for particular flows which satisfy the information level requirements of all applications.

When configuring the firmware, the SDM controller communicates directly with the SW Access module. In order to maintain a proper functionality of SDM firmware, the controller must carry out the following operations:

- Management of rules activated in the firmware (rule add/delete/update) based on the application demands.
- Cyclic export of active flow records computed in the firmware flow cache.
- Allocation of records in the firmware flow cache.

## III. Proof of Concept

This chapter analyzes the proposed concept. It is divided into three sections. The first section proposes several possibly weak points of the SDM concept. The second section presents an analysis of network traffic. The aim of the analysis is to show whether the SDM concept is a sound idea. The third section draws conclusions about the presented analysis and addresses all of the proposed weak points.

### A. Potential Weak Points

From the presented SDM concept one can infer several potential weak spots in the system design. Their existence can (in bad circumstances) lead to lower effectiveness of hardware preprocessing usage and therefore to a low degree of achieved application acceleration. Major recognized potential weaknesses of the SDM design are the following:

- **Long duration of the feedback loop.** In order to maintain a throughput of 100 Gbps and more, the hardware processing of packets cannot wait for software decisions – the packets must be processed on the fly. Therefore, the action chosen for the flow does not affect a certain amount of leading packets from this flow. If a high portion of flows on the monitored link have an extremely short duration, the acceleration ratio achievable from the usage of SDM declines.
- **Limited firmware capacity.** Because of the fine granularity of preprocessing control, the firmware must store some information about each known flow. The capacity of table with search rules or flow records in the firmware (Table1 or Table2 in Fig. 3) can be restrictive. An extremely high number of concurrent flows on the network can restrict the preprocessing usage to only a small portion of the flows. Negative effects of this restriction can be significantly reduced by an adequate selection of preprocessed flows. Suitability of the flow is given by the achievable reduction of its data during preprocessing. It is generally desirable to prefer the preprocessing of large (heavy) flows.
- **Insufficient data reduction.** Hardware preprocessing reduces the data quantity from the network by converting the packets into Unified Headers, aggregating them into flow records or by dropping them completely. The amount of data reduction is directly proportional to the size of processed packets and flows. Therefore, in the case of extremely short flows with very short packets the effectiveness of data reduction of the SDM can be relatively small.
- **Overly granular control.** The choice of the acceleration control basic unit affects the number of required rules in the Search module and the rate of their creation. The benefit from a preprocessing rule covering a small portion of the incoming traffic is small. In the extreme case, the overhead of rule creation can even outweigh the SDM benefits. Also, rule generation in case of extremely small units of control can exceed the achievable throughput of the configuration interface.

### B. Network Traffic Analysis

The magnitude of possible negative impacts of the described weak spots is closely related to the character of processed data. Therefore, we have analyzed the properties of the network traffic in a real high-speed backbone network. Based on the measured characteristics we have proven that the proposed SDM system can perform very well when deployed in real networks.

All of the measurements in this paper were conducted in the high-speed CESNET2 backbone network. CESNET2 is Czech NREN which has optical links operating at speeds up to 100 Gbps and routes mainly IP traffic. We conducted all of our measurements during the standard working hours of the workweek. We measured mean size of packets in bytes, mean size of flows in packets and mean time duration of flows. Because we aim for the application protocols, we measured the mentioned characteristics, not only for the whole network traffic on the link, but also for the selected application protocols. We selected a set of interesting protocols: HTTP, HTTPS, DNS, SMTP, SSH and SIP. Furthermore, we measured

the percentage of these protocols in the captured traffic in the matter of flows, packets and bytes.

The results of the basic network traffic analysis are shown in Table I. The table shows that the statistics vary depending on the application protocol. Dominant is the HTTP protocol with more than a quarter of all flows and more than a half of all packets and bytes. Moreover, HTTP flows and packets are generally larger (heavier) and longer. A considerable amount of traffic belongs to HTTPS, which has generally smaller and longer flows than HTTP. A high amount of flows also belong to the DNS protocol (one fifth), but this number is highly disproportional to the DNS total packet and bytes percentage. DNS flows are generally very small (light). A majority of them consists of only one small packet.

| | Flows [%] | Packets [%] | Bytes [%] | Flow [packets] | Flow [s] | Packet [Bytes] |
|---|---|---|---|---|---|---|
| **HTTP** | 25.45 | 54.36 | 58.68 | 63.1 | 7.167 | 963.2 |
| **HTTPS** | 14.28 | 6.92 | 4.75 | 14.3 | 8.493 | 611.7 |
| **DNS** | 18.89 | 0.72 | 0.17 | 1.1 | 0.179 | 207.2 |
| **SMTP** | 0.38 | 0.22 | 0.14 | 17.2 | 2.934 | 573.8 |
| **SSH** | 0.04 | 0.01 | 0.00 | 11.6 | 17.433 | 233.0 |
| **SIP** | 0.00 | 0.00 | 0.00 | 4.9 | 24.701 | 420.9 |
| **others** | 40.96 | 37.76 | 36.26 | 27.3 | 7.735 | 856.7 |
| **all** | | | | 29.6 | 6.257 | 892.2 |

TABLE I. BASIC STATISTICAL CHARACTERISTICS OF NETWORK DATA GROUPED BY THE APPLICATION PROTOCOL

Another interesting characteristic of the network is the *distribution* of packet lengths. The majority of packets are either very long (over 1300 B: 57 %) or very short (under 100 B: 35 %). Especially dominant are both extremes from the range of lengths supported by the Ethernet standard – 42 and 1500 B. Medium sized packets are not very common.

There is already information about mean flow durations for the selected application protocols in Table I. Further information about the flow time durations can be seen in Fig. 4. Each line in the graph shows the percentage of flows that last shorter than the given duration. Generally (red thicker line) over $\frac{2}{3}$ of all flows are shorter than 100 ms and only a tenth of them exceed a duration of 10 s. Also majority of DNS and SIP flows have a duration under 10 ms.

Fig. 4 shows further information about flow duration, but does not say anything about time distribution of packets inside the flows. Weights of individual flows are also not considered. A better look at packet timing inside the flows can be shown by measuring the relative arrival times of packets from the start of the flow. Thus, the first packet of each flow has the zero relative arrival time and its absolute arrival time marks the starting time of that flow. Then, each consequent packet has a relative arrival time equal to the difference of its absolute arrival time and the marked start of the flow. Results of this measurement are shown in Fig. 5. The graph shows that generally (red thicker line) only a small portion of all packets arrive right after the start of the flow – only a fifth of all packets arrive during the first second of flow. This fact leads to the conclusion that flows with short duration carry only a very few packets. The conclusion is further strengthened by the fact that the majority of flows have a very short duration. There are exceptions such as DNS and SIP though.

There is already information about mean flow sizes for selected application protocols in Table I. Further information

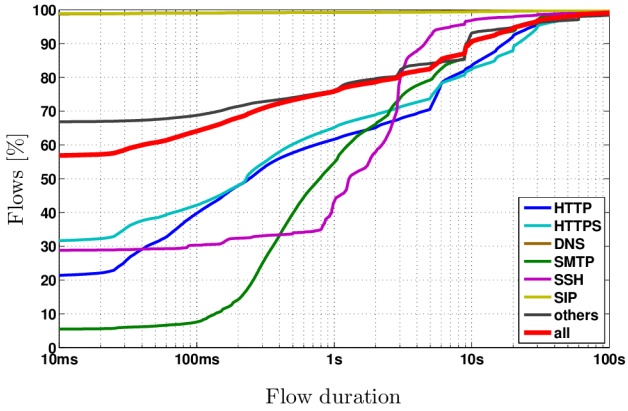


Fig. 4. Cumulative distribution functions of flow durations

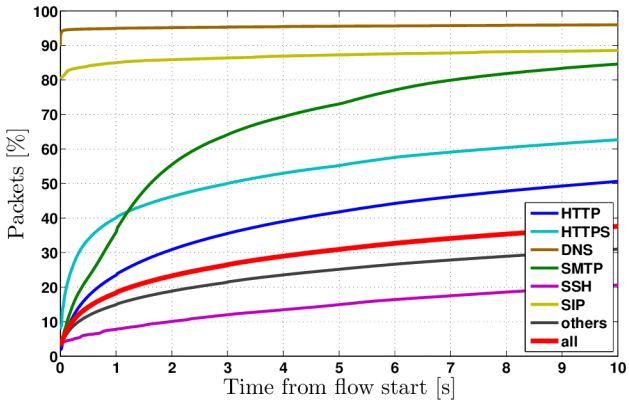


Fig. 5. Cumulative distribution functions of packet arrival times

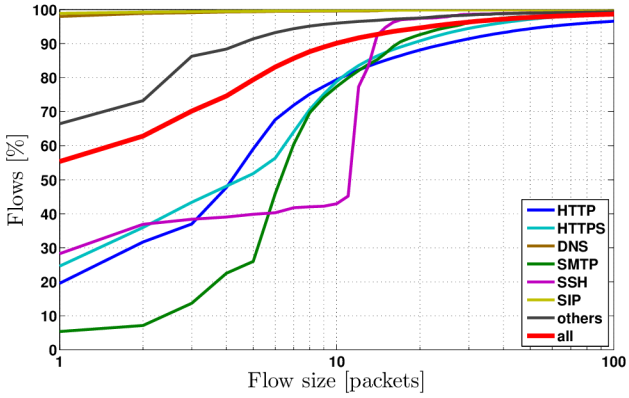


Fig. 6. Cumulative distribution functions of flow sizes

about flow sizes can be seen in Fig. 6. Each line of the graph shows the percentage of flows that consists of less packets than a given number. Generally (red thicker line) only a tenth of all network flows have more than 10 packets. Also, virtually all DNS and SIP flows consist of a single packet.

Fig. 6 shows further information about flow sizes, but does not clearly say anything about the percentage of all packets carried by flows of different sizes. It is known that

high-speed network traffic has a heavy-tailed character of flow size distribution. The heavy-tailed character of flow size distribution derived from the measured values is shown in Fig. 7. The graph shows the portions of all packets carried by the specified percentage of the heaviest flows on the network. It can be seen that generally (red thicker line) 0.1 % of the heaviest flows carries around 60 % of all packets and 1 % carries even around 85 %. An exception to the heavy-tailed distribution of flow sizes is the DNS protocol. On the other hand, SIP and SSH protocols have a heavier tail than average.

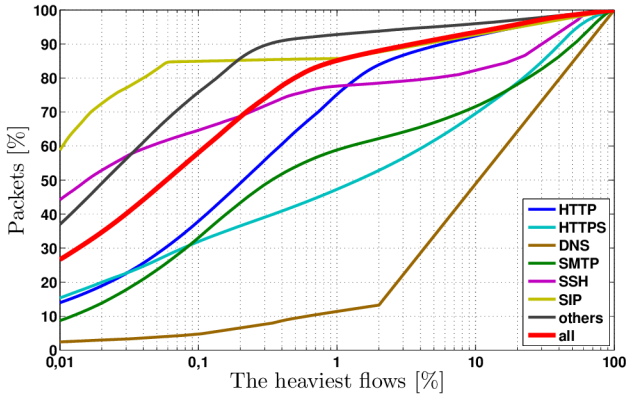


Fig. 7. Portions of packets carried by the percentage of the heaviest flows

A consequence of the heavy-tailed character of the network traffic is that even by selecting a small percentage of the heaviest flows, we can cover the majority of packets. The problem then lies in the effective prediction of which flows are among the heaviest. More accurately, it lies in the capability to recognize the heaviest flows only from the properties of their first few packets. The simplest method of this recognition is based on the rule that every flow is considered heavy after the arrival of its first $k$ packets for some selected decision threshold $k$. The main advantage of this method is its simplicity – no packet analysis nor advanced stateful information for the flows is needed.

The measured accuracy of the heaviest flow selection by the described simple method is shown in Fig. 8 and Fig. 9. These graphs show the relations between the value of threshold $k$ to the portion of heavy marked flows (first graph) and packets covered by them (second graph). By a combination of values from both graphs we can see that with the rising decision threshold the portion of flows marked heavy dramatically decreases, but the percentage of covered packets decreases rather slowly. For example, decision threshold $k = 20$ leads to only 5 % of heavy marked flows covering around 85 % of all packets. Exceptions are the DNS and to some extent also HTTPS and SMTP protocols, where the percentage of covered packets decreases quickly.

A different view of the simple heavy flow prediction method effectiveness can be seen in Fig. 10. It shows the mean number of packets covered by one heavy marked flow for different values of the decision threshold $k$. Values shown in the graph rise with the decision threshold to a considerably higher number than the mean sizes of the flows from Table I – hundreds or even thousands of packets instead of only tens

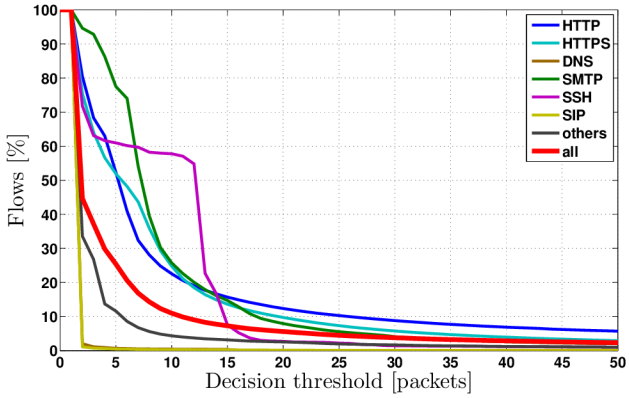


Fig. 8. Heavy flow detection using the simple method – portions of selected flows

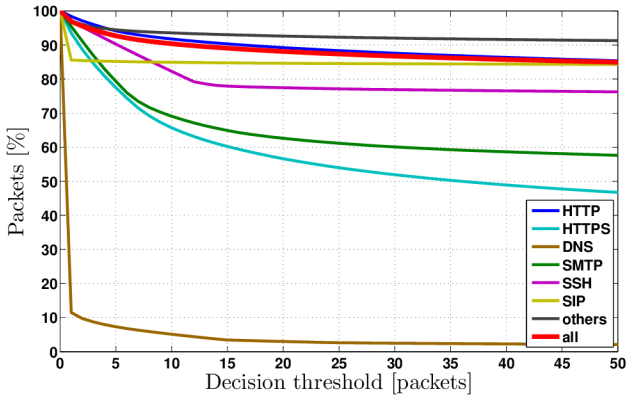


Fig. 9. Heavy flow detection using the simple method – portions of captured packets

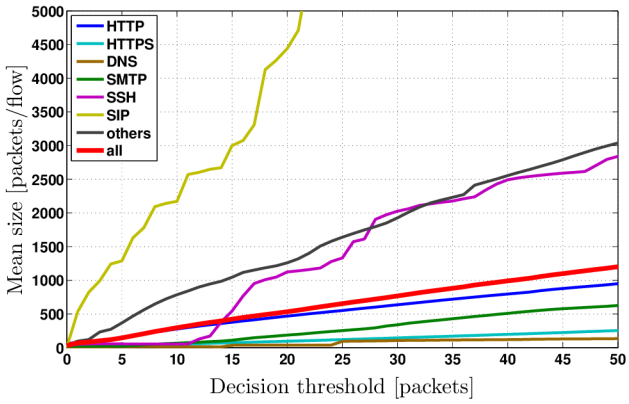


Fig. 10. Mean number of captured packets per flow in flows selected using the simple method

of them. This clearly proves that even a simple heavy flow prediction method effectively predicts the heaviest flows.

### C. Proof of Concept Conclusion

Based on the analysis results presented in this section we can now draw conclusions about the negative effects of

possible weak spots of the SDM design. The conclusions are:

- **Long duration of feedback loop.** The expected SDM feedback loop delay is in the area of tens to hundreds of milliseconds. Fig. 4 shows that the majority of flows has a duration too short for this requirement (over $\frac{2}{3}$ shorter than 100 ms). But in spite of that, the majority of packets is carried by longer flows and arrives later from the flow start (only a tenth of packets during the first 100 ms according to Fig. 5). These results lead to a small negative effect of feedback loop duration on the system performance.
- **Limited firmware capacity.** Fig. 7 shows a heavy-tail character of network traffic. Moreover, figures 8 and 9 show that even a very simple heavy flow prediction method can give very good results. In conclusion, even with a relatively small number of flow rules it is possible to cover the majority of packets.
- **Insufficient data reduction.** Unified Headers and Flow Records have sizes of tens of bytes. Table I shows that rather large packets are mostly used – the mean size is nearly 900 B. Therefore, a reduction of network traffic bytes is sufficient.
- **Overly granular control.** Fig. 10 shows that with an appropriate selection of flows it is possible to achieve a high effectiveness of rules. Each rule can specify a preprocessing offload into HW of hundreds or even thousands of packets on average.

From these conclusions it is clear that possible weak spots of the SDM design will not have a large negative impact on system performance in real networks. Exceptions are protocols like DNS with a very high percentage of single packet flows. Fortunately, these protocols cover only a small portion of network traffic (e.g. DNS with less than 1 %).

## IV. Results

In order to verify the proposed system further, we have implemented the whole SDM system prototype. The hardware part of the system is realized by the accelerator board with the powerful Virtex-7 H580T FPGA. The whole FPGA firmware occupies less than half of the available FPGA resources. That includes not only the SDM functionality, such as packet header parsing and NetFlow statistics updating, but also 100 Gbps Ethernet, PCI-Express and QDR external memory interface controllers. The software is realized as a set of plugins for the Invea-Tech's Flowmon exporter software [4]. This exporter allows us to modify its functionality to the extent required by the SDM system.

The designed SDM system brings acceleration of monitoring applications based mainly on software defined hardware acceleration of network traffic preprocessing. Control of the preprocessing is mainly realized by the monitoring applications through on the fly defined dynamic rules for particular flows. These rules are generated as a reaction to the first few packets of the flow. Therefore, there is some delay between the flow start and rule application. The duration of this delay influences the portion of packets affected by the rules. The basic view of achievable SDM system effectiveness can be gained from an examination of an achievable portion of packets whose preprocessing was influenced by the dynamic flow rules.

In order to test the described ability of the SDM system we created a simple use case. In this use case, only a specified number of the first packets from each flow is interesting to the software. All packets from unknown (new) flows are, therefore, by default forwarded into the software application. SDM controller software counts the number of packets in each flow. Right after the reception of the specified number of packets for a flow, the application creates a rule for the firmware to drop all the following packets from this flow. This decision method is absolutely the same as the simple heavy flow detection method defined in the previous section.

In the described test case we have measured the portion of packets dropped by the SDM firmware. The results are projected into the graph in Fig. 11. The graph shows the percentage of dropped (influenced) packets (solid lines) and the percentage of flows for which the rule was created (dashed lines). For comparison, analytical results from graphs 8 and 9 in the previous section are also shown (red). The result is that the SDM system can influence preprocessing of up to 85 % of all packets from real network traffic by dynamic flow rules. A visible difference of about 10 % of influenced packets between analytical and real results is caused by neglecting the duration of rule creation and activation process in the analytical result.

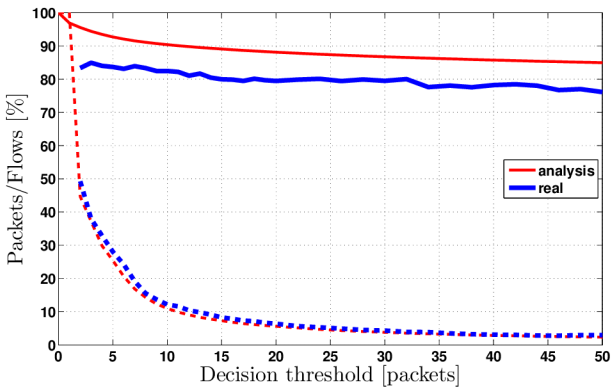


Fig. 11. Portions of offloadable packets and flows using the simple heavy flow detection method

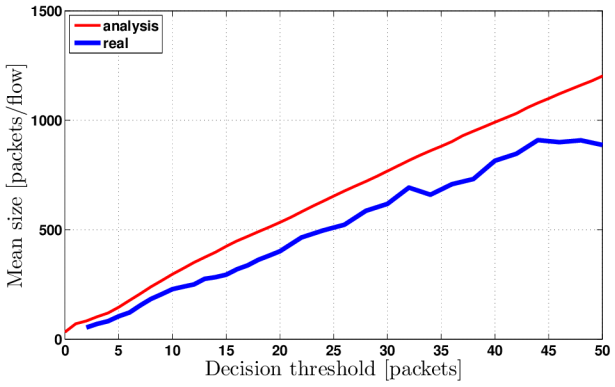


Fig. 12. Mean number of offloadable packets per flow in flows selected using the simple heavy flow detection method

The graph in Fig. 11 also shows a similar character of packets and flows portions as described in the previous section – considerably faster decline in the percentage of flows than in the percentage of packets. A better view is provided in Fig. 12. There, the relation of the mean number of packets influenced by one created rule over the decision threshold value is shown (blue). The red line is analytical result of simple heavy flow detection method effectiveness taken from Fig. 10. The graph shows that real measured effectiveness of this method is slightly worse than the analysis suggests. But it is still very effective and suitable for real usage.

Apart from this artificial use case, we also tested SDM acceleration abilities in more realistic use cases. We tested the performance of the system in the following four cases:

- **Standard NetFlow measurement.** In this use case, all packets from the link are taken into account. By default, they are sent to the software in the form of UH. The software adds dynamic rules to offload the NetFlow measurement of heavy flows (predicted by the simple method) into the hardware accelerator.
- **HTTP header analysis.** We choose HTTP because HTTP traffic is dominant in the networks. Therefore, the acceleration of its analysis is of high importance. In this use case we tested the application that parses HTTP headers and extracts some interesting information (e.g. URL, host, user-agent) from them. Extracted information can then be used to augment the flow records. Because the application works with the data of HTTP packets, only the packets with a source or destination port 80 are sent into the software by default. Others are dropped in the hardware. Furthermore, the application adds dynamic rules to drop the packets of HTTP flows in which it already detected and parsed the HTTP header.
- **Standard NetFlow enriched by HTTP analysis.** This case combines the two previous ones. Both applications are active at the same time without the need of any changes in them. Their traffic requirements are automatically combined by the SDM controller.
- **DNS security analysis.** We choose DNS because it is a bit different from the other protocols. Its flows are extremely short. Therefore, the dynamic flow rules have virtually no effect on DNS preprocessing. But the DNS traffic takes up less than a hundredth of all network traffic. So, even with the use of default rules only (no dynamic rules), SDM should be able to massively accelerate the analysis.

The results of the SDM system testing in the described use cases are shown in Figures 13 and 14. The figures show the portions of all incoming packets and bytes preprocessed in the hardware by a particular method. These hardware preprocessing utilizations lead to a reduction of software application load displayed in Table II. The table shows portions of incoming packets and bytes that are processed by software applications in particular use cases relative to the state without the SDM accelerator. It also shows the percentage of flows for which the rule is created in the hardware.

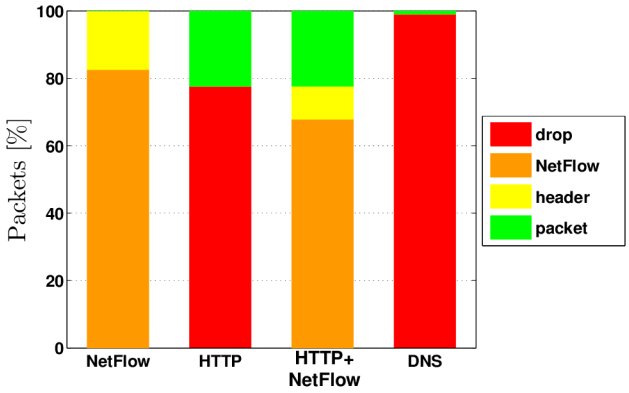


Fig. 13. Portions of hardware preprocessing types in tested use cases

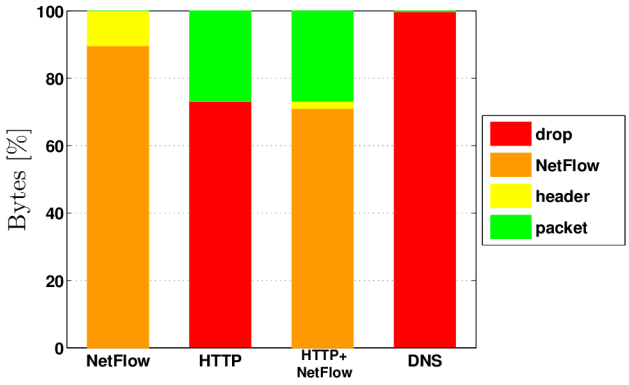


Fig. 14. Portions of hardware preprocessing types in tested use cases

| | SW load [%] | | Rules in HW |
|---|---|---|---|
| | Packets | Bytes | [% of flows] |
| **NetFlow** | 17.55 | 0.86 | 12.16 |
| **HTTP** | 22.32 | 26.85 | 3.60 |
| **HTTP+NetFlow** | 32.42 | 27.30 | 11.84 |
| **DNS** | 0.73 | 0.16 | 0.00 |

TABLE II. SOFTWARE APPLICATIONS LOAD USING SDM IN TESTED USE CASES, RELATIVE TO THE STATE WITHOUT THE SDM ACCELERATOR

Standard NetFlow measurement is mostly accelerated by the hardware flow cache. In this way, the software application load is reduced to less than a fifth of all packets (in the form of UH or flow record). Further acceleration rises from the fact that only UHs and flow records are sent to the software, instead of complete packets. The software, therefore, does not parse packets anymore and the PCI Express load is reduced to less than one percent.

SDM accelerates the analysis of application protocols by packet dropping based on static and dynamic rules. This leads to the HTTP parser load being reduced to only about a fifth of all packets and bytes and to the DNS parser load reduced to less than a hundredth.

When the standard NetFlow measurement and the application protocol parsing are used simultaneously, the load of the application protocol parser is the same as when used alone thanks to the DMA channel traffic splitting supported by the SDM. The HTTP parser software still receives only

the packets on the TCP port 80. The load of the software NetFlow measurement slightly rises compared to the NetFlow only measurement, because of the packets that are sent to the software for the HTTP analysis (NetFlow measurement sees also the HTTP packets).

## V. Related work

We discussed several recent works that may to some extent resemble the SDM concept. We, however, have shown that our work has significant differences with those papers.

The proposed arrangement of our system resembles OpenFlow [5]: Packets of an unknown flow are passed from the data path to the controlling software, which in turn may choose to install processing rules into the data path. Similar to plugins for the OpenFlow controller, SDM is also designed to support various software plugins. The main difference with OpenFlow is that our system is aimed solely at monitoring, with the ability to achieve a great amount of flexibility by using software monitoring plugins. For the sake of performance, the SDM controller is very tightly coupled with the hardware accelerator. There is also an outlook to further improve the system in terms of types of measurements that are performed by the hardware accelerator (besides NetFlow). Therefore, our system is an instance of Software Defined Networking in a broader sense, yet it is completely different from OpenFlow.

FlowSense [6] is a lightweight system aiming at estimating the network performance such as link utilization. It uses the built-in counters of OpenFlow switches to estimate the network parameters. While this approach brings virtually no overhead, its possibilities are limited by the OpenFlow protocol messages content and no other measurement can be done using this technique. There is no support for application level processing in FlowSense.

The OpenSketch architecture [7] defines a configurable pipeline of hashing, classification and counting stages. These stages can be configured to perform the computation of various statistics. OpenSketch is tailored to compute *sketches* – a probabilistic structure allowing us to measure and detect various aspects of the network communication with a defined error rate. It is not intended for hard NetFlow-like monitoring, nor for exact, error-free measurements. Also, OpenSketch does not allow for application level protocol parsing.

The Shunt system [8] is a hardware accelerator with the support to divert a suspicious/interesting traffic to the software for further analysis. To this end it resembles our work, however, Shunt accelerates only packet forwarding and does not include any possibilities to offload/accelerate the flow measurement tasks. Our work is also more complete by defining the software architecture with the plugin support.

## VI. Conclusion

We have designed a new concept of application level flow monitoring acceleration called Software Defined Monitoring. The concept is able to support application level monitoring and high-speed flow measurements at speeds over 100 Gbps at the same time. Our system focuses on high speed and high quality flow based measurement with the support of a hardware accelerator. The accelerator is fully controlled by the software feedback loop and offloads the simple monitoring tasks of bulk, uninteresting traffic. The software, on the other hand, decides about the traffic processing on a per-flow basis and performs the advanced monitoring tasks such as application protocol parsing. The software works with monitoring plugins, therefore, SDM is *by design* ready for extensions by new high-speed monitoring tasks without the need to modify its hardware. It is also anticipated that the hardware accelerator will be improved to support additional types of offload in addition to current packet parsing and NetFlow statistics counting.

We have performed a detailed analysis of the backbone network traffic parameters so as to assess the feasibility of the concept. We have also implemented the whole SDM system using the Virtex-7 FPGA accelerator board. The system is ready to handle 100 Gbps traffic. Using the SDM prototype, we have evaluated several use cases for SDM. It is clear from the obtained results that SDM is able to offload a significant part of the network traffic to the hardware accelerator and therefore to support a much higher throughput than a pure software solution. The results show a major speed-up in all test cases.

## Acknowledgment

This research has been partially supported by the “CESNET Large Infrastructure” project no. LM2010005 funded by the Ministry of Education, Youth and Sports of the Czech Republic, the research programme MSM 0021630528, the grant BUT FIT-S-11-1 and the IT4Innovations Centre of Excellence CZ.1.05/1.1.00/02.0070.

## References

[1] B. Claise, “Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information,” Internet Requests for Comments, Internet Engineering Task Force, RFC 5101, January 2008. [Online]. Available: http://tools.ietf.org/html/rfc5101

[2] T. Martínek and M. Košek, “Netcope: Platform for rapid development of network applications,” in *Design and Diagnostics of Electronic Circuits and Systems, 2008. DDECS 2008. 11th IEEE Workshop on*, 2008, pp. 1–6.

[3] V. Puš, L. Kekely, and J. Kořenek, “Low-latency modular packet header parser for fpga,” in *Proceedings of the eighth ACM/IEEE symposium on Architectures for networking and communications systems*, ser. ANCS ’12. New York, NY, USA: ACM, 2012, pp. 77–78.

[4] INVEA-TECH a.s., “FlowMon Exporter – Community Program,” 2013. [Online]. Available: http://www.invea.cz

[5] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “Openflow: Enabling innovation in campus networks,” *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[6] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, and H. V. Madhyastha, “Flowsense: Monitoring network utilization with zero measurement cost,” in *Proceedings of the 14th international conference on Passive and Active Measurement*, ser. PAM’13. Berlin, Heidelberg: Springer-Verlag, 2013, pp. 31–41.

[7] M. Yu, L. Jose, and R. Miao, “Software defined traffic measurement with opensketch,” in *Proceedings of the 10th USENIX conference on Networked Systems Design and Implementation*, ser. nsdi’13. Berkeley, CA, USA: USENIX Association, 2013, pp. 29–42.

[8] N. Weaver, V. Paxson, and J. M. Gonzalez, “The shunt: An fpga-based accelerator for network intrusion prevention,” in *Proceedings of the 2007 ACM/SIGDA 15th international symposium on Field programmable gate arrays*, ser. FPGA ’07. New York, NY, USA: ACM, 2007, pp. 199–206.

## A.2 Paper 2

**Trade-offs and Progressive Adoption of FPGA Acceleration in Network Traffic Monitoring**

# Trade-offs and Progressive Adoption of FPGA Acceleration in Network Traffic Monitoring

Lukáš Kekely, Viktor Puš, Pavel Benáček
CESNET a. l. e.
Zikova 4, 160 00 Prague, Czech Republic
Email: kekely,pus,benacek@cesnet.cz

Jan Kořenek
IT4Innovations Centre of Excellence
Faculty of Information Technology
Brno University of Technology
Božetěchova 2, 612 66 Brno, Czech Republic
Email: korenek@fit.vutbr.cz

***Abstract*—Current hardware acceleration cores for network traffic processing are often well optimized for one particular task and therefore provide high level of hardware acceleration. But for many applications, such as network traffic monitoring and security, it is also necessary to achieve rapid development cycle to provide fast response to security threats. We propose and evaluate a new concept of hardware acceleration for flexible flow-based network traffic monitoring with support of application protocol analysis. The concept is called Software Defined Monitoring (SDM) and it relies on a configurable hardware accelerator implemented in FPGA, coupled with smart monitoring tasks running as software on general CPU. The monitoring tasks in the software control the level of detail and type of information retained during the hardware processing. This arrangement allows rapid application prototyping in the software, followed by further shifting of the timing critical parts of the processing to the hardware accelerator. The concept is proposed with the scalability in mind, therefore it is suitable for different FPGA based platforms ranging from embedded single-chip solutions (such as Zynq or Cyclone V) to high-speed backbone network monitoring boxes. Our pilot high-speed implementation using FPGA acceleration board in a commodity server performs a 100 Gb/s flow traffic measurement augmented by a selected application protocol analysis.**

## I. Introduction

The task of network traffic monitoring is one of the key concepts in modern network engineering and security. A golden standard in the area of network monitoring is a flow measurement. A monitoring device collects basic statistics about the network flows and reports them to a central storage collector using a handover protocol such as NetFlow [1] or IPFIX[2]. Flow measurement is a stateful process, because for each packet the flow state record is updated in the device (e.g. packet counters are incremented), and only the resulting numbers are exported. The ongoing trend in this field is towards creating richer flow records [3], [4], [5], carrying some extra information in addition to the basic flow size and timing statistics. The added information often include values from the application level protocol headers, such as HTTP, DNS etc.

Implementations of the application level flow monitoring solely in software are certainly possible, yet their throughput is limited mainly by the performance of commodity processors. FPGAs offer much better possibilities in terms of throughput. However, a fixed solely hardware implementation may face the flexibility issues, since the evolving nature of network threats implies the need for fast changes of the monitoring process, quickly making fixed hardware devices obsolete.

The aim of this paper is to (1) strike a balance between the system throughput and flexibility/programmability and to (2) offer a configurable trade-off to the above, but mainly to (3) endorse a progressive adoption of network monitoring subtasks to the hardware accelerator, driven solely by the needs of the networking community.

We employ a hardware accelerator to perform the reduction of traffic for software applications by partial offloading of the packet parsing and flow aggregation into hardware. Therefore, the accelerator passes some of the packets (as requested) intact to the software while performing the flow measurement (or other aggregation) of the bulk traffic that is not interesting to the application-layer processing software tasks.

The use of packet processing offload can be controlled on a per flow basis by the monitoring software and adjusted on the fly according to its actual needs. Offload control is realized through unified interface by a dynamically specified set of flow rules. These rules are installed into the accelerator to determine the type of packet preprocessing acceleration used for individual network flows. The preprocessing method that best aids the performance and does not violate the precision requirement of advanced software processing is selected.

The whole system is designed to be easily extensible at two main levels. At the software side, monitoring plugins can be added to the system. This brings the possibility of rapid development and deployment of new monitoring applications, for example as a reaction to a new network security threat. Once the functionality of software task is verified and stable enough, the second level of the system extensibility can be employed to further speed-up the task. Various packet processing and data aggregation routines can be relocated directly into the hardware accelerator. Furthermore, the system is designed to scale well from small embedded devices up to the 100 Gbps backbone network monitoring boxes.

## II. Analysis

We start the paper with the analysis of the properties of the network traffic in a real high-speed backbone network. All of our measurements were conducted in the high-speed CESNET2 backbone network. This research and educational

network has optical links operating at speeds up to 100 Gbps and routes mainly IP traffic.

The key question for the analysis to answer is how big reduction of data can we achieve by a system based on the following basic concepts: (1) the core of network traffic processing is realized entirely in the software, (2) acceleration is achieved by a software controlled offload of the flow processing to the hardware, based on the few leading packets, (3) target family of applications (monitoring and security) usually does not require most of the network traffic – aggregated information or only a specific fractions of traffic are sufficient.

Very important characteristic of network traffic is the flow size distribution. According to graph derived from the measured values, shown in Fig. 1, the flow size distribution has heavy-tailed character. The graph shows the portions of all packets carried by the specified percentage of the heaviest flows (i.e. flows with the most packets) in the network. It can be seen that generally (black thicker line) 0.1 % of the heaviest flows carry around 60 % of all packets and 1 % carries even around 85 %. A consequence of this observation for the proposed system concept is that even by offloading a small portion of the heaviest flows, we can accelerate the preprocessing of the majority of packets.

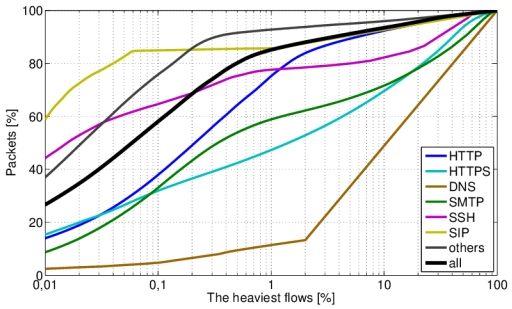


Fig. 1. Portions of packets carried by the percentage of the heaviest flows

The problem then lies in the capability to predict the heaviest flows only from the observed properties of their first few packets. From a wide variety of heavy flow detection methods we choose one that is very simple: For a selected threshold $k$, a flow is considered heavy after the arrival of its first $k$ packets. The main advantage of this method is its straightforward implementation – no deep packet analysis nor advanced stateful information for the flows is needed.

The measured accuracy of the heaviest flow selection by this method is shown in Fig. 2. The graph shows the relation between the value of threshold $k$ and the portion of heavy marked flows (dashed line) and packets (solid line) covered by them. By a combination of values we can see that with the rising decision threshold the portion of heavy marked flows dramatically decreases, but the percentage of covered packets decreases rather slowly.

## III. Architecture

The basic idea behind the acceleration by the proposed SDM system is based on a finely controlled load reduction

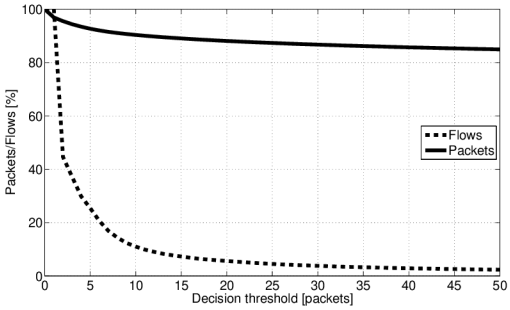


Fig. 2. Heavy flow detection using the simple method – portions of offloaded flows and packets

and distribution achieved by the accelerated preprocessing of the network traffic. Although the preprocessing is done by the firmware in FPGA, it is fully controlled by the software applications. Therefore, the earliest few packets of each new flow are sent to the software, which selects a type of hardware preprocessing used for the subsequent packets of the said flow.

The suitable types of hardware preprocessing for the area of network monitoring can be divided into three basic groups:

- **Extraction** of the interesting data from packets and sending only those data to the software in a fixed format, which we call Unified Header (UH).
- **Aggregation** of packets into flow records directly in the hardware. This aggregation does not need to be only basic flow statistics, but different forms of aggregation can be specified according to the needs of particular applications.
- **Filtration** of unnecessary packets and forwarding only the interesting ones into the software. This can aid advanced monitoring applications, which perform various analyses and detections oriented only to some specific subgroup of network traffic.

The top-level conceptual scheme of the proposed SDM system is shown in Fig. 3. The processing of an incoming packet in the FPGA firmware starts with the header parsing and extraction of packet metadata (Parser). Extracted metadata is then used to classify the packet based on a software defined set of rules (Rule Lookup). Each rule identifies one concrete flow and specifies the type of packet preprocessing and the target software channel for packets of that flow. Packets can be processed in a firmware flow cache (i.e. aggregated to selected type of flow record), dropped or sent to the software unchanged or in the form of Unified Header.

The data from the firmware is sent over the bus to the software using multiple independent channels. Data for each channel is stored in a software buffer in the form of whole packets, Unified Headers or flow records.

This data is processed by the set of user specific software applications such as the flow exporter [1] which analyzes the received data and exports the flow records to the collector. User applications read the data from the selected channels. They also specify which types of traffic they want to inspect and which flows can be preprocessed in hardware. Definitions

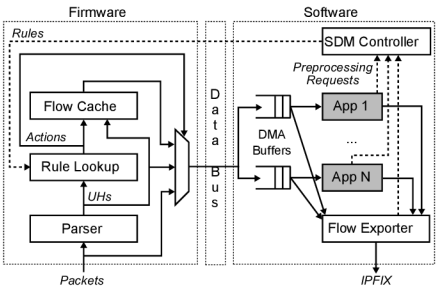


Fig. 3. Conceptual top-level scheme of SDM system

of (un)interesting traffic are passed from all applications to the software SDM Controller. The SDM controller aggregates the definitions (requests) into rules and configures the firmware preprocessing in order to achieve the maximal possible reduction of the traffic while preserving the required level of information. The network traffic preprocessing in the firmware is entirely controlled from the software and the core of the controlling software are the monitoring applications. Each monitoring application has the form of an SDM plugin. The main input to the plugin is the data path carrying the packets, extracted UHs or aggregated flow records. The plugin output is whichever data that the plugin has parsed/detected/measured. This output data can be added to the exported IPFIX flow record, so that it is *enriched* by the information from the plugin. The third interface of the monitoring application is the flow (dis)interest information interface to the SDM Controller. SDM controller accepts the preprocessing requests from multiple applications and aggregates them into rules for the firmware. This mechanism realizes the feedback control loop, which is an important concept in our work.

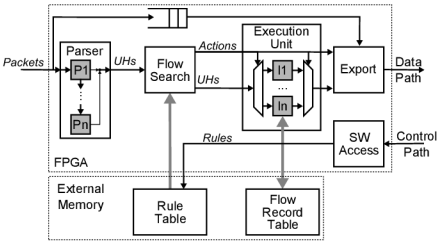


Fig. 4. Detailed firmware scheme

Fig. 4 shows a top level implementation scheme of the SDM accelerator firmware for FPGA. The main firmware functionality is realized by the processing pipeline which processes the incoming network traffic and creates an outgoing data flow for the software. The SDM firmware is realized by five main modules:

**Parser** extracts interesting information from headers of packets, especially fields that clearly identify network flows. To identify the flows, we use the 5-tuple: IP addresses, TCP/UDP ports and protocol. Furthermore, our implementation is modular and enables easy extensions of default packet parsing process by additional application-specific parser modules (P1..Pn).

**Flow Search** assigns an action (processing instruction) to every packet based on its flow identifier and a set software defined rules. Management of the rule set is done through a control interface capable of an atomic on the fly add, remove or update of the rules.

**Execution Unit** manages the stateful flow records in Flow Record Table. It mainly actualizes their values by execution of instructions from flow associated actions. Every action specifies an instruction to be executed and the address of the flow record to work with. Furthermore, the instruction has access to data extracted from packet (UH). The Execution Unit supports multiple user-defined instruction sub-modules (I1..In), more details about the execution and implementation of instructions are in Sec. III-A.

**Export** pairs together corresponding UH transaction with frame data from FIFO buffer. Then it chooses the required channel and format for the data based on action assigned by the Flow Search module.

**SW Access** is the main access point into the SDM firmware from the software side. Its primary function is to manage the rules and to initiate the export of the flow records based on controller commands.

### A. *Execution Unit functionality*

Execution Unit realizes the main stateful behavior of the hardware by execution of flow record updating instructions. To improve the overall flexibility of the system, we use modular architecture that allows to implement custom read-modify-write aggregation operations (instructions). Thanks to these custom instructions, the nature of the flow records maintained by the hardware in Flow Record Table can be customized according to the target application. We use high-level synthesis (HLS) tools to generate custom hardware modules from the description in C or C++. Thanks to that, SDM hardware can be customized faster and even without the knowledge of HDL programming (e.g. by network security experts).

We implement and evaluate five different Execution Unit instructions to test the feasibility of the described concept:

- **NetFlow** instruction is used for standard NetFlow aggregation. Its execution increases flow packet and byte counters, updates flow end timestamp and computes logical OR of the observed TCP flags.
- **NetFlow Extended** instruction has the same basic functionality as NetFlow. In addition, it stores the TCP flags of the first five packets.
- **TCP Flag Counters** instruction performs increment of counters of individual observed TCP flags. For example, one can see the number of ACK flags transmitted during the whole TCP connection.
- **Timestamp Diff** instruction maintains records of inter-arrival times of the first eleven packets of the flow.
- **CPD** instruction represents the Change-Point Detection algorithm [6], [7] designed to detect an anomaly in the processed network flow.

## IV. Results

We implement and evaluate a high-speed version of SDM system. We realize the hardware part by the PCI Express accelerator board with the Virtex-7 H580T FPGA.

### A. Achieved performance

As we describe earlier, the designed SDM system accelerates the monitoring applications by the software defined hardware acceleration of network traffic preprocessing. The preprocessing control is realized by the monitoring applications through on the fly defined dynamic rules for particular flows. There is some delay between the flow start and rule application in the FPGA firmware. The duration of this feedback loop delay can influence the portion of packets affected by the rules (i.e. offloaded by the hardware accelerator).

Therefore, we measure the portion of packets that were processed by the SDM firmware according to selected flow rules. The results are shown in Fig. 5. The graph shows that, the measured effectiveness of the system (red) is only slightly worse than the analysis (black from Fig. 2) suggests. The gap is only from 5 to 10 % of all packets for our implementation. The width of the gap between the theoretical and practical results can be further reduced by utilization of a platform with shorter latency than that of PCI Express (e.g. CPU core(s) and FPGA logic within the same chip).

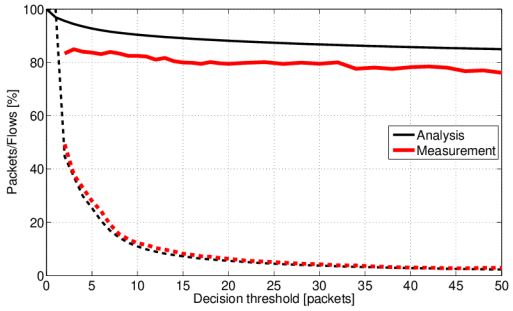


Fig. 5. Portions of offloadable packets and flows using the simple heavy flow detection method

### B. FPGA implementation results

Our high-speed SDM FPGA firmware runs at 200 MHz and includes not only the SDM core functionality, as described in Sec. III, but also Ethernet, PCI-Express and QDR external memory interface controllers. Closer look at the FPGA resources of the firmware is shown in Tab. I. Using the same SDM core with the data width of 512 bits and throughput of 100 Gbps, we create 3 different FPGA architectures for boards with 3 different arrangements of Ethernet ports: one 100 GbE port, two 40 GbE ports and eight 10 GbE ports.

Table II shows the resource utilization of the individual instruction sub-modules for the Execution Unit. It can be seen that the additional instruction sub-modules are relatively small, compared to the whole firmware, and therefore adding new instruction should not involve any major refinements of the FPGA firmware.

TABLE I
Resources of the SDM firmware

| Throughput | Regs | LUTs |
|---|---|---|
| 1×100 Gbps | 197 758 | 249 214 |
| 2×40 Gbps | 134 172 | 178 984 |
| 8×10 Gbps | 184 084 | 222 745 |

TABLE II
Resources of the instruction blocks

| Instruction | Regs | LUTs |
|---|---|---|
| NetFlow | 1846 | 824 |
| NetFlow Extended | 2070 | 1113 |
| TCP Flag Counters | 0 | 1046 |
| Timestamp Diff | 5199 | 2556 |
| Change-Point Detection | 5296 | 3919 |

## V. Conclusion

Our work shows the design and implementation of a flexible 100 Gb/s network flow monitoring system working at the application layer using a commodity PC and a hardware accelerator. The behavior of the system is fully controlled by the software, which makes us use the term Software Defined Monitoring. The concept is by design extensible by the software plugins to adjust its functionality to actual needs and to react to future network threats. Next level of extensibility is provided by custom instructions of the hardware accelerator, which can redefine the nature of the acceleration itself.

## Acknowledgement

This research has been partially supported by the "CESNET Large Infrastructure" project no. LM2010005 funded by the Ministry of Education, Youth and Sports of the Czech Republic, the project TA03010561, the research programme MSM 0021630528, the grant BUT FIT-S-11-1 and the IT4-Innovations Centre of Excellence CZ.1.05/1.1.00/02.0070.

## References

[1] B. Claise, "Cisco Systems NetFlow Services Export Version 9," RFC 3954, Internet Engineering Task Force, October 2004.

[2] B. Claise, B. Trammell, and P. Aitken, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information," RFC 7011, Internet Engineering Task Force, Sept. 2013.

[3] L. Deri, L. Trombacchi, M. Martinelli, and D. Vannozzi, "A distributed dns traffic monitoring system," in *8th International Wireless Communications and Mobile Computing Conference (IWCMC)*, 2012, pp. 30–35.

[4] M. Elich, P. Velan, T. Jirsik, and P. Celeda, "An investigation into teredo and 6to4 transition mechanisms: Traffic analysis," in *38th Conference on Local Computer Networks Workshops*, 2013, pp. 1018–1024.

[5] P. Velan, T. Jirsík, and P. Čeleda, "Design and evaluation of http protocol parsers for ipfix measurement," in *Advances in Communication Networking*, ser. Lecture Notes in Computer Science, T. Bauschert, Ed. Springer Berlin Heidelberg, 2013, vol. 8115, pp. 136–147.

[6] A. Tartakovsky, A. Polunchenko, and G. Sokolov, "Efficient computer network anomaly detection by changepoint detection methods," *Selected Topics in Signal Processing*, vol. 7, no. 1, pp. 4–11, 2013.

[7] R. B. Blazek, H. Kim, B. Rozovskii, and A. Tartakovsky, "A novel approach to detection of "denial–of–service" attacks via adaptive sequential and batch–sequential change–point detection methods," in *Proc. 2nd IEEE Workshop on Systems, Man, and Cybernetics*, 2001.

## A.3 Paper 3

**FPGA Accelerated Change-Point Detection Method for 100Gb/s Networks**

# FPGA Accelerated Change-Point Detection Method for 100 Gb/s Networks

Tomáš Čejka[1], Lukáš Kekely[1], Pavel Benáček[2], Rudolf B. Blažek[2], and Hana Kubátová[2]

[1] CESNET a. l. e.
Zikova 4, Prague, CZ
cejkat,kekely@cesnet.cz
[2] CTU in Prague – FIT
Thakurova 9, Prague, CZ
benacekp,rblazek,hana.kubatova@fit.cvut.cz

**Abstract.** The aim of this paper is a hardware realization of a statistical anomaly detection method as a part of high-speed monitoring probe for computer networks. The sequential Non-Parametric Cumulative Sum (NP-CUSUM) procedure is the detection method of our choice and we use an FPGA based accelerator card as the target platform. For rapid detection algorithm development, a high-level synthesis (HLS) approach is applied. Furthermore, we combine HLS with the usage of Software Defined Monitoring (SDM) framework on the monitoring probe, which enables easy deployment of various hardware-accelerated monitoring applications into high-speed networks. Our implementation of NP-CUSUM algorithm serves as hardware plug-in for SDM and realizes the detection of network attacks and anomalies directly in FPGA. Additionally, the parallel nature of the FPGA technology allows us to realize multiple different detections simultaneously without any losses in throughput. Our experimental results show the feasibility of HLS and SDM combination for effective realization of traffic analysis and anomaly detection in networks with speeds up to 100 Gb/s.

## 1 Introduction

Computer networks are getting larger and faster, and hence the volume of data captured by network monitoring systems increases. Therefore, there is a need to analyze more data for detection of network attacks and traffic anomalies. This paper deals with real-time detection of attacks suitable for high-speed computer networks thanks to the direct deployment of detection methods in hardware monitoring probe.

Today, monitoring systems usually consist of several probes that capture and preprocess huge amounts of network traffic at wire speed, and one or more collector servers that collect and store network traffic information from these probes. Analysis of network data is traditionally also realized at the collectors. In this

paper, we propose a different approach, where anomaly detection is shifted directly into the monitoring probes. The aim of this approach is to enable real-time analysis even in very large networks with speeds up to 100 Gb/s per Ethernet port and to reduce the latency of anomaly detections.

It is virtually impossible to process all network data from the 100 Gb/s link in software using only commodity hardware. The main limitations lay in insufficient bandwidth of communication paths between the network interface card and the software components [1] and in limited performance of the processors. Therefore, hardware acceleration must be used for high-speed networks in order to avoid transferring and processing of all the data in the software.

In this paper, we utilize a special network interface card mounted with FPGA chip for hardware acceleration of network traffic processing as a basis for our high-speed probe. The FPGA on the card allows us to realize more advanced data processing features (e.g. anomaly detection methods that use packet level statistics) directly on the card, thus reducing the data load for the software. To demonstrate this approach, we concentrate on a real-time sequential Change-Point Detection (CPD) method that is designed to minimize the average detection delay (ADD) for a prescribed false alarm rate (FAR) [2,3].

As the basis for the FPGA firmware, we use Software Defined Monitoring (SDM). SDM is a novel monitoring approach proposed in [4], that can be used as a framework for hardware acceleration of various monitoring methods. SDM combines hardware and software modules into a tightly bound co-design that is able to address challenges of monitoring from data link to application layer of the ISO/OSI model in modern network environments at the speeds up to 100 Gb/s.

The main contribution of this paper is the evaluation of a statistical real-time detection methods implemented in hardware. The detection methods are extensions of a hardware accelerated monitoring probe designed for 40 Gb/s and 100 Gb/s Ethernet lines. The resulting device is able to analyze unsampled high-speed network traffic without loss.

The rest of this paper is organized in the following way. Introduction to the implemented sequential non-parametric change-point detection method (NP-CUSUM) can be found in Sec. 2. The used SDM concept is briefly described in Sec. 3. Sec. 4 describes created hardware implementation of detection method. Evaluation of the developed system and the achieved results are presented in Sec. 5. Related work and main differences between existing projects and our implementation are presented in Sec. 6. Sec. 7 summarizes the results presented in this paper and outlines our future work.

## 2 Change-Point Detection

Network attacks, intrusions, or anomalies appear usually at unpredictable points in time. The start of an attack is mostly observable as a change of some statistical properties of the network traffic or its specific part. Therefore, methods based on sequential Change-Point Detection theory are suitable for intrusion detection. CPD methods detect the point in time where the distribution of some

perpetually observed variables changes. In network security settings, these variables correspond to some relevant, directly observed or calculated network traffic characteristics. The main problem of such approach is the lack of precise knowledge about the statistical distributions of these traffic characteristics. Ideally, the distributions should be known both, before and after the distribution change that corresponds to the anomaly or attack. Therefore, we use a non-parametric CPD method NP-CUSUM that was developed in [2,3] and that does not require precise knowledge about these statistical distributions.

NP-CUSUM is inspired by Page's CUSUM algorithm that is proven to be optimal for detection of a change in the mean (expectation) when the distributions of the observed random variables are known before and after the change [5]. The typical optimality criterion in CPD is to minimize the average detection delay (ADD) among all algorithms whose average false alert rate (FAR) is below a prescribed low level. Page's CUSUM procedure, which is based on the log-likelihood ratio, can for i.i.d. (independent and identically distributed) random variables $X_n$ be rewritten [5] as:

$$U_n = \max\left\{0, U_{n-1} + \log\frac{p_1(X_n)}{p_0(X_n)}\right\}, \quad U_0 = 0, \tag{1}$$

Where $p_0$ and $p_1$ are the densities of $X_n$ before and after the change, respectively.

The formulation in (1) is the inspiration for the NP-CUSUM method procedure [2,3]. The procedure is applicable to non–i.i.d. data with unknown distributions (i.e. the method is non-parametric). First, the Page's CUSUM procedure was generalized as $S_n = \max\{0, S_{n-1} + f(X_n)\}$ with some function $f$. Changes in the mean value of $X_n$ can be detected using sequential statistic:

$$S_n = \max\{0, S_{n-1} + X_n - \hat{\mu} - \varepsilon\hat{\theta}\}, \quad S_0 = 0, \tag{2}$$

Where $\hat{\mu}$ is an estimate of the mean of $X_n$ before the attack, $\hat{\theta}$ is an estimate of the mean after the attack started, and $\varepsilon$ is a tuning parameter for optimization. It has been shown in [3] that with optimal value of $\varepsilon$ the NP-CUSUM procedure (2) is asymptotically optimal as FAR decreases. That is, for small prescribed rate of false alarms, other procedures will have longer detection delays. In fact, the delays can theoretically be exponentially worse [3].

As the input of the NP-CUSUM algorithm, we can use various features $X_n$ of the observed network traffic. To basically evaluate our hardware implementation of the method, we have chosen for $X_n$ the ratio of SYN and FIN packets of the Transmission Control Protocol (TCP) in a short time window [6]. During "normal" operation of the network, each connection is opened using two SYN packets, and closed using two FIN packets (one in each direction). Therefore, we expect the ratio of SYN and FIN packets to be on average close to 1 or at least constant. Sudden and consistent change of the ratio is suspicious and can be caused by some sort of attacks (e.g. SYN or FIN packet flood) [6].

To demonstrate the scalability and power of our hardware implementation using SDM, we raise the number of observed statistics and add some more NP-CUSUM blocks in parallel. The added statistics utilize information about ICMP

and RST TCP packets. All measured values are used in form of ratios in order to avoid the dependency on trends and traffic volumes that could increase the number of false alerts. Finally, thanks to parallelism, observation of multiple statistics simultaneously does not negatively affect the processing throughput.

## 3 Software Defined Monitoring System

Software Defined Monitoring (presented in [4,7]) forms a basis for our hardware implementation of detection methods in a monitoring probe. In this section we briefly describe the main architecture of the SDM system and the changes needed to accommodate the implementation of NP-CUSUM monitoring system.

An SDM system consists of two main parts: firmware for the FPGA on hardware accelerator and software for general processors. The hardware and software components are connected via a PCI-Express bus. Both parts are tightly coupled together to allow precise software control of hardware processing. The software part of the SDM system consists of monitoring applications and a controller. The monitoring applications can perform advanced monitoring tasks (such as analysis of application protocols) or also export information (alerts) to the collector. The controller manages the hardware module by dynamically removing and inserting processing rules into its memory (see Fig. 1). The instructions contained in the rules tell the hardware what actions to perform for each input packet with some characteristics. These rules are defined by the monitoring *Applications*, which inserts them to the *Hardware* via the *Controller*.

Due to aforementioned facts, the monitoring application can not only use data coming from the hardware, but it can also manage the details of hardware processing of network traffic as well. The offloading of traffic processing into the hardware saves both, the bandwidth of communication interface (PCIe) and the CPU processing time. The hardware module can pass information to the software in the form packet metadata from a single packet, or as aggregated records computed from multiple subsequent packets with common features (such as NetFlow [8] aggregation). Whole received packets or their parts can be also sent to the software for further (deeper) analysis. Graphical representation of the SDM concept is shown in Fig. 1.

Processing of an incoming network packet in the SDM hardware starts with the extraction of its protocol headers. The extracted data are used to search adequate rule in memory that specifies the desired processing possibly supplemented by address of a record. The selected rule and metadata for each given packet are then passed to the SDM *Update* block, which is the heart of the SDM concept making that idea strong. This block contains a routing table that is used to forward the incoming processing request to the appropriate update (instruction) blocks, for execution. Each of these instruction blocks can perform a specific update operation (realize a specific aggregation type) on the record. Each update operation is delimited by two memory operations: reading the stored record values, and writing back the updated values. Also, new types of updates (aggregations) can be specified, simply by implementing the new instruction block

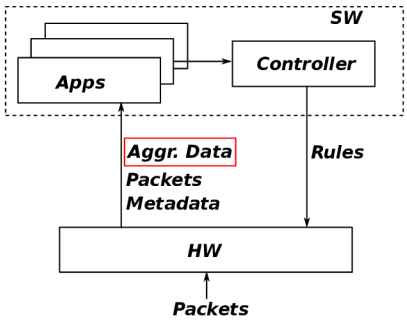


**Fig. 1.** Software Defined Monitoring (SDM) abstract architecture

and plugging it into the existing *Update* block infrastructure. A special type of processing action is an export into the software of the processed packet data, metadata, or stored values from a selected record, optionally followed by clearing of that record. Records can be exported when some special condition is met or in periodical manner.

## 4 Implementation

### 4.1 The CPD hardware block

Our hardware implementation of CPD method is realized as hardware plug-in for the SDM system. More precisely, it is available as a new instruction block for the SDM *Update* module that is described in the previous section. The SDM design supports access to arbitrary data records stored in memory for instruction blocks. Although, the available data size of a record is limited due to memory block size that can be read or written on each clock cycle – the block size is equal to 288 b. Usage of bigger data records than 288 b would cause unwanted latency increase and lower throughput of the whole monitoring hardware.

One CPD instruction block uses available space in memory to store: previous historical value, 2 parameters of the NP-CUSUM algorithm, and 1 threshold value that is used for alerting purposes. Memory should also contain counters with observed features such as the number of SYN or FIN packets, and the packet counter that starts the ratio and NP-CUSUM computation. The data stored in memory is accessible from software and therefore all of the thresholds and parameters can be changed on the fly.

The source code of the instruction block allows us to specify the data type size of all values stored in memory. The choice of data type sizes implies the number of hardware blocks that can work in parallel in the same clock cycle with the same memory block. However, the decrease of data type size lowers the value precision and data ranges. The NP-CUSUM parameters, the previous historical value and the threshold are represented as 16 b decimal numbers. The counters are set to 8 b. For one block that analyzes SYN/FIN ratio, the implementation

works with 88 b of memory for one record in total. Configuration with 4 NP-CUSUM blocks uses 5 counters (SYN, FIN, RST, ICMP, packet counter) and 4 sets of fixed-point values. In total, 4 NP-CUSUM blocks would use 296 b of memory. Therefore the size of decimal number data type was shortened to 15 b and the total used memory size was decreased to available 280 b.

We use a high-level synthesis (HLS) approach [9], to implement the CPD method from Sec. 2 for the FPGA as an instruction block inside the SDM system. The structure of the implemented block is shown at Fig. 2. The main advantage of using HLS approach is faster implementation of new hardware accelerated monitoring and detection methods with minimal loss of efficiency in comparison to traditional coding of FPGA firmware using Hardware Description Languages (HDL) such as VHDL or Verilog. Following the requirements for the SDM instruction block interfaces and general behavior, we have developed the CPD hardware block in the C++ language.

Implementation of the CPD hardware block brings a several issues to solve. The most important one is the choice of decimal numbers representation. We try two of the standard approaches: fixed-point and floating-point representation. The main advantage of the floating-point approach is the ability to represent a greater range of values. But on the other hand, hardware realization of floating-point arithmetic is very complicated and considerably slower. Therefore, the usage of fixed-point arithmetic can be favored by better performance and lower resource usage of the instruction block.

From the HLS point of view, the most important parameter for our design goals is the achievable Initiation Interval (II). This parameter represents the number of clock cycles needed for initialization of a new request in the instruction block. Ideally, we require the II to be equal to one so that a new request can be accepted in each clock cycle and the instruction block is able to achieve full throughput. During our experiments, we have discovered that the effect of decimal numbers representation on the II is following: floating-point version of the instruction block has II of 11 clock cycles whereas the fixed-point version has II of 1.

Another very important performance-related parameter of our implementation is latency. It is required to be as small as possible because high latency can lead to delays between repeated processing of the same instruction caused by the fact that records in the memory need to be locked in order to achieve atomic processing. In the end, our experimental timing and performance results indicate that the created implementation is able to handle network traffic at 100 Gb/s Ethernet line. More detailed results regarding our synthesis and FPGA requirements are discussed in Sec. 5.

Apart from CPD instruction block creation, another important part of the implementation is connection of the new instruction block to the existing SDM *Update* block. Thanks to the by design extensibility of SDM *Update* block, this task is simple and straightforward. All that needs to be done is to wrap the translated HLS implementation of the new block in a VHDL envelope that is responsible for adapting the behavior of all predefined interface signals. The

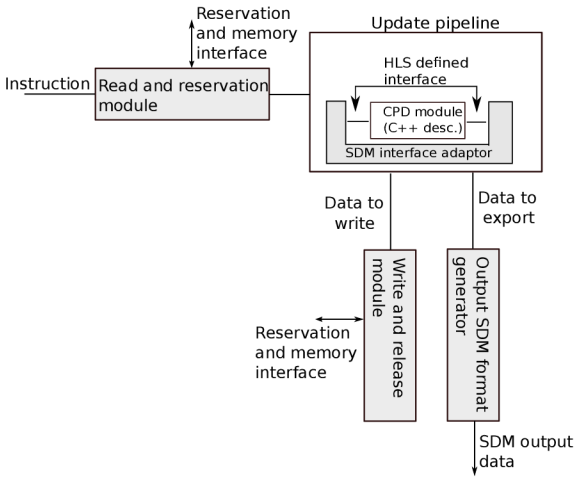


**Fig. 2.** Implementation of the CPD Instruction

wrapping process is depicted in Fig. 2. The gray blocks are parts of the SDM designated for connecting of a new instruction blocks. The SDM can thus be viewed as some kind of a framework that brings the possibility to create new hardware modules for rapid network monitoring acceleration.

To finish the implementation of the Change-Point Detection method in the SDM system, a software monitoring application needs to be created. The application communicates with an SDM Controller daemon to manage the detection details in hardware module (see Fig. 1) and also receives detected alerts. The main task of the monitoring application is to control the detection process and present its results to human operators.

## 5 Evaluation

Correct functionality of the created implementation of the CPD block was verified using referential software application. The referential application is written in the plain C language and is not meant to be highly optimized for the HLS. Its main purpose is only to validate the functionality of the hardware implementation. In addition, the software application is extended and serves as the base for the measuring and detection application [10] that can be used in slower networks or for estimation of configurable parameters for the CPD block.

We have implemented the hardware-based prototype of the NP-CUSUM detection method as an instruction block for the SDM Update block in an SDM monitoring probe. The prototype is developed for the network interface card with a 100 Gb/s Ethernet port and a Virtex-7 H580T FPGA, which is the main core for the implemented detection functionality.

A detailed list of all FPGA resources needed for the implementation of one CPD instruction block, which observes one feature, is shown in Tab. 1. In the

table there are also results for other constellations of the CPD blocks that contain more computational blocks with 1, 2, or 4 instances of the NP-CUSUM algorithm and observe more features in parallel. The total number of available resources on used chip is 725 600 Flip-Flops (FF) and 362 800 Look-up tables (LUT). The number of utilized LUTs and FFs for CPD instruction block itself, therefore, accounts only for less than 1% of the available FPGA resources.

**Table 1.** FPGA resources used for the CPD instruction block in different configurations.

| **Name** | 1 block | | 2 blocks | | 4 blocks | |
|---|---|---|---|---|---|---|
| | FF | LUTs | FF | LUTs | FF | LUTs |
| Expression | 0 | 458 | 0 | 496 | 0 | 479 |
| Instance | 280 | 252 | 560 | 504 | 560 | 504 |
| Multiplexer | - | 1842 | - | 1868 | - | 2130 |
| Register | 2253 | - | 2377 | - | 2593 | - |
| ShiftMemory | 0 | 806 | 0 | 816 | 0 | 814 |
| Total | 2533 | 3358 | 2937 | 3684 | 3178 | 3982 |

Performance results for the CPD instruction blocks are shown in Tab. 2 and Tab. 3, whereas Tab. 3 shows detailed information about the fixed-point implementation. An Initiation Interval is required to be equal to one in order to support processing of 100 Gb/s network traffic at full wire-speed (see Sec 3). This requirement is not satisfied only by the floating-point implementation. Vivado HLS version 2013.2 was used for high-level C to VHDL synthesis. Xilinx ISE version 14.7 with enabled synthesis optimization was used for VHDL to FPGA netlist synthesis. Enabling the optimization such as register duplication leads to a higher clock frequency achieved for the final implementation and also to a higher resources consumption. The tables illustrate that after the optimization all performance requirements from Sec. 3 have been met by the fixed-point implementation.

**Table 2.** Comparison of timing results for the synthesized CPD instruction blocks.

| **Parameter** | **Reached Fixed-point** | **Reached Floating-point** | **Required** |
|---|---|---|---|
| Clock period | 4.08 ns | 16.48 ns | 5 ns |
| Frequency | 245 MHz | 60.679 MHz | 200 MHz |
| Latency | 12 | 11 | - |
| Initiation Interval | 1 | 12 | 1 |
| Bus Width | 512 b | 512 b | 512 b |
| Achieved Throughput | 125 Gb/s | 2.5 Gb/s | 100 Gb/s |

**Table 3.** Performance results for the CPD instruction blocks in different configurations.

| **Parameter** | **Reached 1 block** | **Reached 2 blocks** | **Reached 4 blocks** | **Required** |
|---|---|---|---|---|
| Clock period | 4.08 ns | 4.20 ns | 4.20 ns | 5 ns |
| Frequency | 245 MHz | 238 MHz | 238 MHz | 200 MHz |
| Latency | 12 | 12 | 12 | - |
| Initiation Interval | 1 | 1 | 1 | 1 |
| Bus Width | 512 b | 512 b | 512 b | 512 b |
| Achieved Throughput | 125 Gb/s | 121 Gb/s | 121 Gb/s | 100 Gb/s |

Finally, Tab. 4 shows the total number of FPGA resources required for the whole synthesized SDM system with one CPD hardware plug-in. The table shows that about 87% of the Virtex-7 H580T resources are still available. Therefore, it is feasible to include several CPD hardware plug-ins in the SDM system for parallel detection of various anomalies without significant latency increase nor throughput loss.

**Table 4.** FPGA resources of the SDM system with one CPD hardware plug-in ( FPGA xc7vh580thcg1155-2).

| Resource Name | Used Resources [-] | Utilization Percentage |
|---|---|---|
| LUTs | 47731 | 13 % |
| Registers | 21089 | 2 % |
| BRAMS | 107 | 11 % |

## 6 Related Work

We present a brief overview of related work with regard to the differences of our work. This section can be divided into two main domains. The first domain is related to the hardware accelerated detectors and the second domain is related to the detection methods. From the hardware point of view, there are two interesting projects somehow similar to our – Gorilla and Snabb Switch.

The Gorilla project [11] is the closest comparable solutions that we found. 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. In addition, the resource consumption of Gorilla is higher than our solution.

The Snabb Switch project [12] shows different approach of network packets processing. This approach uses modified drivers for faster transfer of network

packets from the network interface card to computer's memory. Transferred data are then processed by network applications. There is also available a Snabb Lab with an accessible platform for measuring. This platform consists of the Supermicro motherboard with dual Xeon-E5 and 20x10 GbE (Intel 82599ES) network cards. This configuration allows to process network traffic at speed of 200 Gb/s. Massive usage of this platform is complicated due to large number of network cards. Our solution is able to process network traffic at speed of 100 Gb/s on one Ethernet line (2 ports allows to achieve 200 Gb/s). Our work is focused on full hardware acceleration of network traffic processing using the only one 100 Gb/s Ethernet port.

From the detection method point of view, there are various existing approaches of anomaly detection from many authors. Detection of SYN flood attacks have been studied and well described in many papers. However, this issue is currently still relevant because of increase of network traffic volumes. Detection based on NP-CUSUM is used in [13] by Wang et al., where the authors present their observation about SYN-FIN pairs in network traffic under normal condition: (1) there is a strong positive correlation between the SYN and RST packets; (2) the difference between the number of SYN and FIN packets is close to the number of RST packets. The authors bring experimental evaluation of flood detection using NP-CUSUM, however they mention a possible disadvantage of aggregated counting of packets that can be spoofed by emission of mixed packet types by attacker.

Siris et al. in [14] compare a straightforward adaptive threshold algorithm, which can bring satisfactory performance for attacks with high intensity, and algorithm base on cumulative sum (CUSUM). Adaptive threshold algorithm uses a difference from moving average value computed e.g. by EWMA algorithm. An alarm is signalized when measured value is higher then moving average in last $k$ consecutive intervals. The CUSUM variant of detection algorithm is influenced by seasonality and trends of network traffic (weekly and daily variations, trends and time correlations). The authors propose to use some prediction method to remove non-stationary behavior before applying the CUSUM algorithm. However, because of time-consuming calculations with minor gains compared to simpler approaches, the authors used simpler approach based on application of CUSUM on difference between measured value and result of Exponential Weighted Moving Average (EWMA) [15] algorithm.

Smoothing of the data signal is important for minimizing the number of false alarms that can be caused by high peaks in data. Therefore, the data are usually preprocessed to avoid short-time deviations to detect long-time anomalies. There are various approaches to smooth the signal and the possible way is to exploit some prediction method such as Moving average, EWMA, Holt-Winters [16], or Box-Jenkins (ARIMA) [17] methods. However, dependency of an algorithm on historical and current measured values can be dangerous and can lead to overlooking of an attack. The issue of self-learning and self-adaptive approach is being studied in our current and future work, however, it is out of the scope of this paper.

Salem et al. presented the currently used methods of the network anomaly detection in [18]. The paper evaluates the usage of extended NP-CUSUM called Multi-chart NP-CUSUM, proposed by Tartakovsky et al. in [19], in combination with Count Min Sketch and Multi-Layer Reversible Sketch (sketching method is proposed eg. in [20]) for data aggregation and anomaly detection.

This paper is focused on the hardware implementation of the detection method, whereas other authors usually more or less rely on software processing of aggregated data. Our solution allows the detection method to be real-time and independent on overloaded software part of system.

## 7 Conclusions

In this paper we present implementation and evaluation of the CPD algorithm (NP-CUSUM) as hardware plug-in for the Software Defined Monitoring system. We achieve easy and rapid development of detection hardware blocks for the FPGA thanks to the usage of high-level synthesis. Also, creation of monitoring probe utilizing newly implemented detection method is very simple and straight forward thanks to the utilization of SDM as the platform for high-speed packet processing. Moreover, we show frequency and FPGA resource evaluation of the hardware implementation for the Virtex-7 H580T FPGA, which is large enough and fast enough to accommodate complex network processing.

Results presented in this paper show that our implementation of NP-CUSUM is capable of processing network traffic at the speed up to 100 Gb/s. The firmware of the whole monitoring probe consumes only 13 % of the available resources of the target FPGA and thus leaves space for several additional CPD (NP-CUSUM) hardware plug-ins that can be used for parallel detection of multiple kinds of network anomalies concurrently. In addition, other existing detection methods can potentially be easily implemented in the similar way – as hardware SDM plug-ins for detection of abrupt changes of network traffic characteristics. The limiting factor for deploying detection hardware plug-ins into a monitoring probe is the consumption of FPGA resources. Generally, detection methods with low data storage requirements can be fully implemented as a hardware plug-ins. Moreover, SDM allows creation of hardware-software co-design where only the most critical parts of the more complex detection algorithm can be accelerated. This partially hardware-accelerated approach can reduce the FPGA resource requirements of advanced detection methods with moderate performance loss.

## Acknowledgment

This work is partially supported by the “CESNET Large Infrastructure” project no. LM2010005 funded by the Ministry of Education, Youth and Sports of the Czech Republic and the project TA03010561 funded by the Technology Agency of the Czech Republic.

## References


1. Santiago del Rio, P.M., Rossi, D., Gringoli, F., Nava, L., Salgarelli, L., Aracil, J.: Wire-speed statistical classification of network traffic on commodity hardware. In: Proceedings of the 2012 ACM Conference on Internet Measurement Conference. IMC '12, New York, NY, USA, ACM (2012) 65–72
2. Blažek, R.B., Kim, H., Rozovskii, B., Tartakovsky, A.: A novel approach to detection of "denial–of–service" attacks via adaptive sequential and batch–sequential change–point detection methods. In: Proc. 2nd IEEE Workshop on Systems, Man, and Cybernetics, West Point, NY. (2001)
3. Tartakovsky, A.G., Rozovskii, B.L., Blažek, R., Kim, H.: A novel approach to detection of intrusions in computer networks via adaptive sequential and batch-sequential change-point detection methods. IEEE TRANSACTIONS ON SIGNAL PROCESSING **54**(9) (2006) 3372–3382
4. Kekely, L., Puš, V., Kořenek, J.: Software defined monitoring of application protocols. In: INFOCOM 2014. The 33rd Annual IEEE International Conference on Computer Communications. (2014) 1725–1733
5. Page, E.S.: Continuous inspection schemes. Biometrika **41**(1/2) (1954) 100–115
6. Wang, H., Zhang, D., Shin, K.: Detecting syn flooding attacks. In: INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE. Volume 3. (2002) 1530–1539
7. Puš, V.: Monitoring of application protocols in 40/100gb networks. In: Campus Network Monitoring and Security Workshop, Prague, CZ, CESNET (2014)
8. Claise, B.: Cisco Systems NetFlow Services Export Version 9. RFC 3954 (2004)
9. Feist, T.: Vivado design suite. White Paper (2012)
10. Čejka, T.: Fast TCP Flood Detector. `http://ddd.fit.cvut.cz/prj/FTFD` (2014)
11. Lavasani, M., Dennison, L., Chiou, D.: Compiling high throughput network processors. In: Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays. FPGA '12, New York, NY, USA, ACM (2012) 87–96
12. Gorrie, L.: Snabb switch. `http://www.snabb.co` (2014)
13. Wang, H., Zhang, D., Shin, K.: Detecting syn flooding attacks. In: INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE. Volume 3. (2002) 1530–1539
14. Siris, V.A., Papagalou, F.: Application of anomaly detection algorithms for detecting SYN flooding attacks. Computer communications **29**(9) (2006) 1433–1442
15. Ye, N., Borror, C., Zhang, Y.: Ewma techniques for computer intrusion detection through anomalous changes in event intensity. Quality and Reliability Engineering International **18**(6) (2002) 443–451
16. Brutlag, J.D.: Aberrant behavior detection in time series for network monitoring. In: LISA. (2000) 139–146
17. Box, G., Jenkins, G., Reinsel, G.: Time Series Analysis: Forecasting and Control. Wiley Series in Probability and Statistics. Wiley (2013)
18. Salem, O., Vaton, S., Gravey, A.: A scalable, efficient and informative approach for anomaly-based intrusion detection systems: theory and practice. International Journal of Network Management **20**(5) (2010) 271–293 00019.
19. Tartakovsky, A.G., Rozovskii, B.L., Blažek, R.B., Kim, H.: Detection of intrusions in information systems by sequential change-point methods. Statistical Methodology **3**(3) (2006) 252–293
20. Krishnamurthy, B., Sen, S., Zhang, Y., Chen, Y.: Sketch-based change detection: methods, evaluation, and applications. In: Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement, ACM (2003) 234–247

## A.4 Paper 4

**Design Methodology of Configurable High Performance Packet Parser for FPGA**

# Design Methodology of Configurable High Performance Packet Parser for FPGA

Viktor Puš, Lukáš Kekely
CESNET a. l. e.
Zikova 4, 160 00 Prague, Czech Republic
Email: pus,kekely@cesnet.cz

Jan Kořenek
IT4Innovations Centre of Excellence
Faculty of Information Technology
Brno University of Technology
Božetěchova 2, 612 66 Brno, Czech Republic
Email: korenek@fit.vutbr.cz

***Abstract*—Packet parsing is among basic operations that are performed at all points of a network infrastructure. Modern networks impose challenging requirements on the performance and configurability of packet parsing modules. However, high-speed parsers often use a significant amount of hardware resources. We propose a novel architecture of a pipelined packet parser for FPGA, which offers low latency in addition to high throughput (over 100 Gb/s). Moreover, the latency, throughput and chip area can be finely tuned to fit the needs of a particular application. The parser is hand-optimized thanks to a direct implementation in VHDL, yet the structure is uniform and easily extensible for new protocols.**

***Keywords*—*Packet Parsing; Latency; FPGA***

## I. Introduction

Since computer networks evolve both in terms of speed and diversity of protocols, there is still a need for packet parsing modules at all points of the infrastructure. This is true not only in the public Internet, but also in closed, application-specific networks. There are very different expectations on packet parsers. For example, consider a multi-million dollar business of low-latency algorithmic trading. In this area, the *latency*, which has long been rather neglected parameter, suddenly becomes more important than the raw throughput. Small embedded devices, on the other hand, often require a parser to be very small (in terms of the memory and chip area), yet still to support a rather extensive set of protocols.

With a rising interest in the Software Defined Networking, it is expected that the ”ossification” of networks will be on decline, and new protocols will appear at even faster rate than before. This expectation handicaps fixed ASIC parsers and favours programmable solutions: CPUs, NPUs and FPGAs. Our work focuses on FPGAs, because of their great potential in high-speed networks.

Current high-speed FPGA-based parsers can achieve a raw throughput of over 400 Gb/s at the cost of the extreme pipelining, which increases both the latency and the chip area (FPGA resources) significantly [1]. Also, the configurability issue is solved only partially. Configuring a set of supported protocols is often addressed by a higher-level protocol description followed by an automatic code generation, but the configuration of the implementation details is left unnoticed.

This paper not only presents a novel packet parser design, but also motivates engineers to create a parametrized solutions, demonstrates the need for a thorough exploration of the space of the solutions and suggests several capabilities that a High-Level Synthesis system should possess to succeed in this area.

The paper is organized as follows: Section II introduces several prior published works in this area, Section III describes our implementation of a modular parser design, Section IV lists all the necessary steps to create own parser in our methodology, Section V presents obtained results and Section VI concludes the work.

## II. Related Work

Rather outdated work by Braun et al. [2] uses the onion-like structure of hand-written protocol wrappers to parse packets. However, due to the 32-bits-wide data path and an old FPGA, the parser achieves a throughput of only 2.6 Gb/s. There is no extensive concept of a common interface for module reuse, and it is unclear how the parser scales for a wider data path.

Kobierský et al. [3] describe the packet headers in XML and generate finite state machines, which parse the described protocol stack. However, the number of states in FSMs rises rapidly with the width of the data bus. Also, the crossbar used in the field extraction unit does not scale well.

While not directly related to our work, there has been an extensive research of general High-Level Synthesis systems, usually translating pure or modified imperative languages (such as C, C++, Java) into the hardware. Most of this research aims to find a potential parallelism hidden in the program loops and to make use of it by unrolling the loops and pipelining the computation. However, the *for* or *while* cycle is far from a convenient description of a packet parser, whose most natural model of computation is perhaps a directed graph of mutually dependent memory accesses. That may be the reason why we do not see many results of a general HLS in this area.

There is a general HLS result that was given by Dedek et al. in [4]. Handel-C language is used to describe the design, but the details are not disclosed. The reported speed of 1 454 Mb/s implies that a rather narrow data bus (probably 16 bit) was used. Therefore, the concern is about the scalability in terms of both an effective description in Handel-C and an effective compilation to hardware for much wider data words. This work also demonstrates that using processors for the packet parsing

978-1-4799-4558-0/14/$31.00 ©2014 IEEE

gives poor results. Compared to the Handel-C implementation, a custom RISC processor designed specifically for the packet parsing yields roughly the same chip area, but achieves only a half of the throughput. Using the MicroBlaze [5] processor (which is not optimized for the packet parsing) requires double resources and brings only 5.7 % throughput compared to the Handel-C solution.

A good example of a domain-specific HLS was given by Attig and Brebner in [1]. They utilize their own Packet Parsing (PP) language to describe (with the syntactic sugar of an object orientation) the structure of packet headers and the methods which define parsing rules. The description is then compiled from PP to the pipeline stages implementation. However, the results indicate that the price for a convenient design entry is the chip area and the latency – most parsers with 1024-bit datapath use over 10 % of the resource-abundant Xilinx Virtex-7 870HT FPGA [6] and the latency varies from 292 to 540 ns.

The Kangaroo system [7] uses RAM to store the packets and employs the on-chip CAM to perform a lookahead. Lookahead is the process of loading several fields from the packet memory at once, allowing to parse several packet headers in a single cycle. The dynamic programming algorithm is used to precompute data structures, so that the parsing of the longest paths in a parse tree is the most accelerated by the lookahead, as it is impractical to perform the lookahead for all the possible protocol combinations. This approach has the architectural limitation of storing the packets in the memory and accessing them afterwards. The memory soon becomes a bottleneck. Our approach, however, parses packets ”on the fly”, which means that the only packet data storage are the pipeline registers.

## III. Modular Parser Design

### A. Input Packet Interface

We start with the design of an input packet interface, which conveys packets into (and through) the parser. While the interface design may seem trivial, it becomes very important for high bandwidth applications. This is due to the fact that FPGAs achieve rather low frequency, roughly between 100-400 MHz. To support the bandwidth over 100 Gb/s, we must use a very wide data bus (up to 2048 bits). Since the shortest Ethernet frame is 64 Bytes (512 bits), packet aliasing and aligning become an issue. Therefore, the achievable effective bandwidth is considerably smaller than the theoretical raw bandwidth.

We propose and our packet convey protocol uses two techniques to utilize the raw bandwidth more effectively than the standard approach:

- **Partially aligned start.** The first packet byte may appear at any position aligned to eight bytes. This corresponds to the 40 and 100 Gb/s Ethernet standard. For a data bus wider than 64 bits (8 bytes), this technique allows the packet to start at other positions than the first byte of a data bus word.
- **Shared words.** One data word may contain the last bytes of the packet $x$ and the first bytes of the packet $x + 1$. The packets may not overlap within the word and the partially aligned start condition may not be violated.

Examples of both aforementioned techniques are shown in the Fig. 1. Using these techniques we bring the effective throughput for the usual packet length distribution much closer to the theoretical limit.

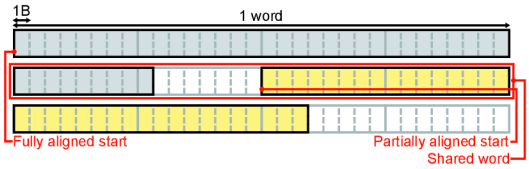


Fig. 1. The example of possible packet positions when using the proposed techniques for the better raw bandwidth utilization.

### B. Parser Submodules

Since we realize that the development of VHDL modules is rather low-level and often very time-consuming, we continue with the design of *Generic Protocol Parser Interface* (GPPI). This interface provides the input information necessary to parse a single protocol header: (1) current packet data being transferred at the data bus, (2) current offset of the packet data and (3) offset of the protocol header. GPPI output information includes (4) extracted packet header field values and the information needed to parse the next protocol header: (5) offset and (6) type of the next protocol header. Fig. 2 shows how the modules are connected. By manually adhering to GPPI, we achieve a hint of object orientation in VHDL – all protocol header parsers use the same interface (except for the extracted header fields) and therefore can easily be interchanged if needed. This improves the code maintainability and enables the easy extensibility of the parser: any new protocol header parser is connected just in the same way as the others. This feature also allows an automatic connection of protocol header parsers from the high-level structure description.

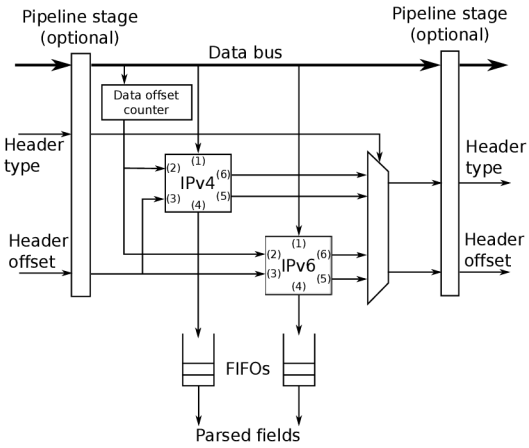


Fig. 2. Example of one pipeline stage.

The inner implementation of each protocol header parser is protocol-specific, but the basic parser block `getbyte` remains the same. This block performs waiting for a specific header field to appear at the data bus, i.e. $po + fo \in \langle do; do + dw \rangle$,

where $po$ is the protocol header offset (module input), $fo$ is the field offset (from the protocol specification), $do$ is the data bus offset (module input), and $dw$ is the data bus width. Once the header field is observed at the data bus, it is stored and can be used to compute the length of the current header, decode the type of the next header, or any other operation. Fig. 3 shows the structure of an IPv4 parser as an example.

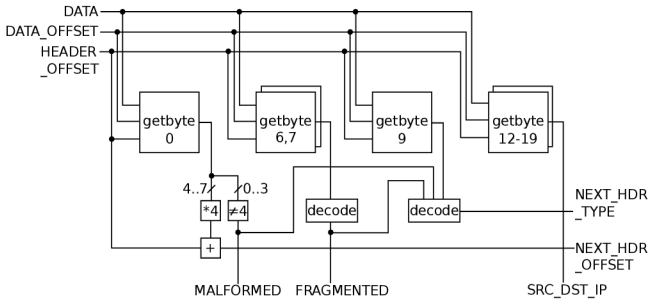


Fig. 3. Example of IPv4 protocol parser.

### C. Parser Top Level

Our parser can output the information about types and offsets of protocol headers. This information is more general than just having the parsed header field values. Obtaining the header field values can be done later, externally to the parser. Our parser offers an option to skip the actual multiplexing of header field values from the data stream. This may save considerable amount of logic resources and is particularly useful for applications that read only a small number of header fields, or when packets are modified in a write-only manner.

Similarly to [1], our parser also uses pipelining to achieve high throughput. However, every pipeline step in our design is optional. If many pipelines are enabled, then the frequency (and the throughput) rises, but also the latency and used logic resources increase. By tuning the use of pipelines, designer can find the optimal parameters for the particular use case.

Each protocol parser contains an inner bypass for situations when its protocol is not present in a packet (not shown in Fig. 3). Thanks to this bypass, the protocol parser submodules can be arranged in a simple pipeline with a constant latency. This property also makes adding a support for new protocols into the parser stack very easy, without the requirement for any changes in the existing protocol parsers. Fig. 4 shows the example top level structure of the parser. Note that the inner bypasses allow to skip certain protocol headers (e.g. VLAN, MPLS), if these are not present in the packet.

The data width required for high throughput (over 100 Gb/s) may be 1024 or even more bits. This implies that there may be more packets in one data word. Our parser is able to handle such situation, provided that no two packet headers of the same type from different packets are present in one data word. For example, if the data word contains the IPv4 header (and the following bytes) of the packet A, and a part of the packet B that includes the IPv4 header, then the packet B is delayed by one cycle in our parser. This situation may only occur only for wide data buses (512 bits and more), and short packets (close to minimal length of 64 bytes) with very

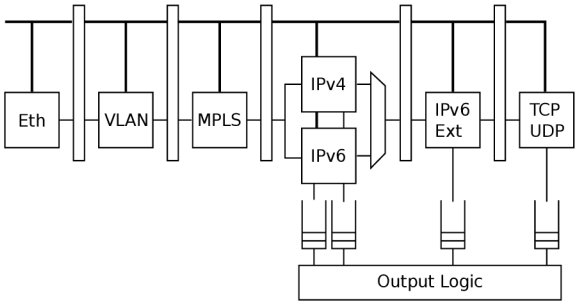


Fig. 4. Example of the parser top level structure.

short inter-packet gap. Our measurements of the real high-speed networks show that it is very rare situation.

## IV. Creating the Parser According to the Application Requirements

With the description of the parser design, it is rather straightforward to create own, customized parser. We identify three basic steps:

- Parser submodules implementation
- Parser top level connection
- Parser state space search

**Parser submodules implementation** comprises manual writing of the VHDL code for each supported protocol. However, GPPI enables easy reuse of the submodules – once written, the protocol parser submodule can always be reused. Also, many generic building blocks of the submodules are already available, for example the `getbyte` module, which extracts a single byte at a certain offset from the packet. In general, the parser submodules for the common protocols are very similar and follow the same informal code template. For many today's protocols the parser submodule is only the `getbyte` modules with correctly configured offsets and some protocol-specific logic to compute the information about the next protocol from the extracted fields. Therefore, one can easily create a parser submodule implementation from the protocol structure specification.

There is also a space here for manual optimizations. For example, extracting a byte from the data bus using the getbyte normally requires a full multiplexer, which is able to extract a byte from any byte position in the data word (Fig. 5a). The multiplexer is controlled by the current data bus offset, the offset of a header within the packet, and the offset of the desired field within the header (which is often constant). However, given the fact that a packet may start only at certain positions in the data word, the current data offset may contain only the values with the corresponding resolution. Also, we can often derive all the possible offsets of the packet header from the analysis of all the possible orderings and sizes of the protocol headers appearing in the packet *before* the current protocol header. Combining the three values (data offset resolution, possible header offsets, constant field offset) together, we can use simpler multiplexers, which do not allow to extract fields from impossible positions. The use of simpler multiplexers in getbyte, together with the fact that getbyte

modules form the main core of the protocol analyzing and data extracting, result in significant chip area savings. For example in a classical TCP/IP protocol stack, header lengths are multiples of 4. Therefore, the size of multiplexers can be reduced 4 times and the size of the whole parsing logic by nearly the same amount. This is illustrated in the Fig. 5.

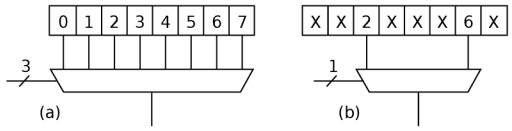


Fig. 5. Example of 64b getbyte multiplexer: full (a) and optimized (b).

**Parser top level connection** once again requires the designer to write VHDL. In this case, the protocol submodules are connected via GPPI pipelines to the structure corresponding to the expected order of the protocol headers in packets. Extracted header field values can be stored in output FIFOs.

**Parser state space search** is the final step. It takes into account other parser requirements than a set of supported protocols. The state space is created by the selective bypassing of pipelines and by setting the data width of the packet convey protocol (all easily set by generic parameters).

For example, there is often a requirement on the throughput. In that case, we are looking for a parser with throughput equal or higher than the requirement. By synthetizing a parser with all the possible settings and ruling out those which do not satisfy the throughput requirement, we obtain a set of satisfying solutions. However, the solutions will differ in the size of chip area and in latency. From this set we select a Pareto set, which contains only the dominating solutions (those for which there is no better solution in both chip area and latency). If the Pareto set has more than one member solution, we have to decide which parameter (area or latency) is more important for our application.

Generally, each candidate solution creates one point in the 3-D space with dimensions throughput, area and latency. Each pipeline step and each data width option double this space, possibly ending in a situation when the exhaustive search is no longer possible, taking into account that a single synthesis run takes time in the order of minutes. In that case we suppose that some global optimization algorithm, such as simulated annealing or a genetic algorithm can be used. Good heuristic helping these algorithms could be to rule out some of the pipeline positions, more precisely to place the pipelines evenly in the parser to create evenly long critical paths.

### A. Implications for High Level Synthesis

After identifying the steps needed to be performed manually, we can now provide a list of features desirable for a good HLS, general or platform specific:

- Way to describe parser interface and protocols.
- Way to specify header formats and their dependency.
- Automatic inference of logical constraints (for multiplexer simplification etc.).
- Generator of parametrized code.
- Way to describe the design goals (area, latency etc.).
- The best fitting solution finder (exhaustive/heuristic).

Note that these requirements do not imply any particular *type* of a parser. Such HLS may generate pipelined parsers similar to ours, or the parsers based on a completely different paradigm (e.g. FSM or processor+code).

## V. Results

We have implemented a parser supporting the following protocol stack: Ethernet, up to two VLAN headers, up to two MPLS headers, IPv4 or IPv6 (with up to two extension headers), TCP or UDP. (see Fig. 6). The parser is able to extract the classical quintuple: IP addresses, protocol, port numbers. Apart from that, it can also provide the information about present protocol headers and their offsets including the payload offset.

We have tested properties of the designed parser with 3 different protocol stacks:

- **full** – Ethernet, 2×VLAN, 2×MPLS, IPv4/IPv6 (with 2× extension headers), TCP/UDP
- **IPv4 only** – Ethernet, 2×VLAN, 2×MPLS, IPv4, TCP/UDP
- **simple L2** – Ethernet, IPv4/IPv6 (with 2×extension headers), TCP/UDP

For each mentioned protocol stack, one test case is done for the parser with the logic to extract the classical quintuple and one for the same parser without the extraction logic (providing only the offsets).

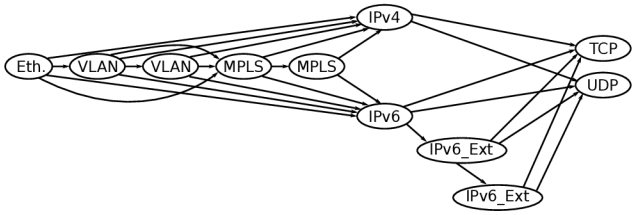


Fig. 6. Structure of supported protocols (full protocol stack).

We provide the results after a synthesis for the Xilinx Virtex-7 870HT FPGA, with different settings of the data width and the number of pipeline stages. These settings, together with the resulting frequency, latency and resource usage, generate a large space of solutions, in which the Pareto set can be found and used to pick the best-fitting solution for an application. In each test case, we use 5 different data widths: numbers from 128 to 2 048 bits that are powers of 2. For each data width, every possible placement of pipelines for the tested protocol stack is shown as a point in the graph and the Pareto set is highlighted. Points representing results for each data width are shown in different shapes and colors.

For each test case we provide 2 graphs: the first one shows the relation between throughput and FPGA resources with the Pareto set highlighted, without any regard to latency. The second graph shows the relation between throughput and latency with the Pareto set highlighted, without any regard

to FPGA resources. In the graph with the relation between throughput and FPGA resources, the second Pareto set (the lower, dashed curve) is also shown. This Pareto set shows the best achievable solutions for our parser without the quintuple extraction logic. Similar Pareto set is not shown in the graph with the relation between throughput and latency, because the usage of the quintuple extraction logic affects the latency of the parser only slightly (the critical paths are mostly in the next header computation logic).

Fig. 7 shows the throughput and the FPGA resources and Fig. 8 shows the throughput and the latency for the full protocol stack. There are 9 configurable pipeline positions in the parser implementing the full protocol stack. This leads to 512 different possible placements of pipelines in this parser for each data width. Mentioned graphs therefore show results for 2 560 different solutions with the Pareto sets highlighted.

For a comparison of the achieved Pareto set results for different protocol stacks, we provide graphs in Fig. 9 (throughput and FPGA resources) and the Fig. 10 (throughput and latency). From these figures one can clearly see that the supported protocol stack can rapidly change the parameters of the parser in terms of chip area and latency. Therefore, a careful protocol support selection is very important for the optimal result. For example, just by turning off the IPv6 support we can bring down the resource utilization by almost 50 %. Latency, on the other hand, is sensitive to the depth of the protocol stack, (see Fig. 6) therefore turning off the support for the VLAN and MPLS headers lowers the latency significantly.

A closer look at the Pareto set optimized for latency and throughput (without regard to FPGA resources) from Fig. 8 is presented in Tab. I. The last line of the table is the estimation of the parser from [1] with similar configuration of the supported protocols (TcpIP4andIP6). It is obvious that our parser can achieve much better parameters than the parser from [1].

| Data Width | Pipes | Throughput [Gb/s] | Latency [ns] | LUT-FF pairs |
|---|---|---|---|---|
| 256 | 0 | 14.5 | 17.1 | 3 238 |
| 512 | 0 | 28.4 | 18.0 | 4 053 |
| 2 048 | 0 | 96.9 | 21.1 | 17 685 |
| 2 048 | 1 | 158.5 | 25.9 | 18 547 |
| 2 048 | 2 | 212.8 | 28.9 | 18 317 |
| 2 048 | 4 | 333.0 | 30.8 | 21 775 |
| 2 048 | 5 | 352.0 | 34.9 | 22 373 |
| 2 048 | 7 | 453.0 | 36.2 | 26 728 |
| 2 048 | 8 | 478.1 | 38.6 | 29 301 |
| 1 024 | ? | 325 | 309 | 67 902 |

TABLE I. PARETO SET FOR THE BEST THROUGHPUT AND LATENCY OF THE FULL PROTOCOL STACK PARSER

Next, we provide the data for the example from the Section IV: Given a set of supported protocols and the target throughput, find all solutions in the Pareto set. We use three sets of supported protocols mentioned earlier and the target throughputs of 40, 100 and 400 Gb/s. All nine Pareto sets are shown in the Fig. 11. Note that while there are several solutions with the throughput over 400 Gb/s, there is only one 400 Gb/s Pareto solution for each protocol set, which means that the other solutions are not better in terms of FPGA resources nor latency. For the other target throughputs, the designer can choose the appropriate solution according to application priorities.

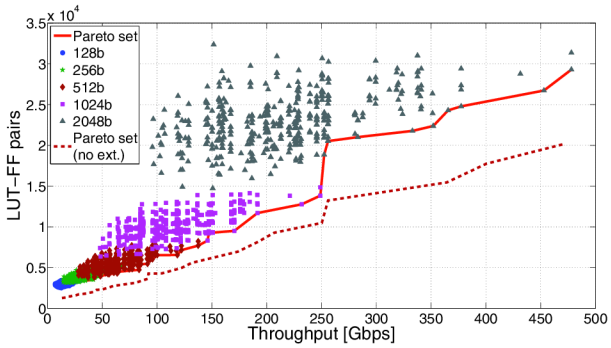


Fig. 7. The FPGA resource utilization for different settings of the full parser.

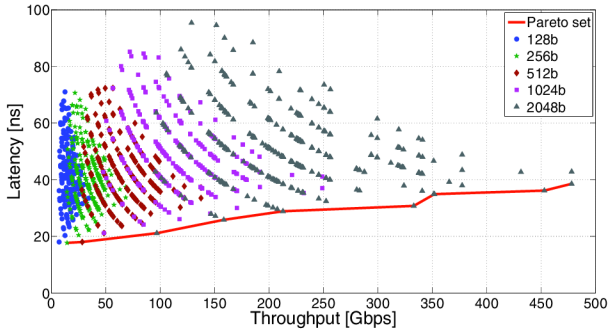


Fig. 8. The latency for different settings of the full parser.

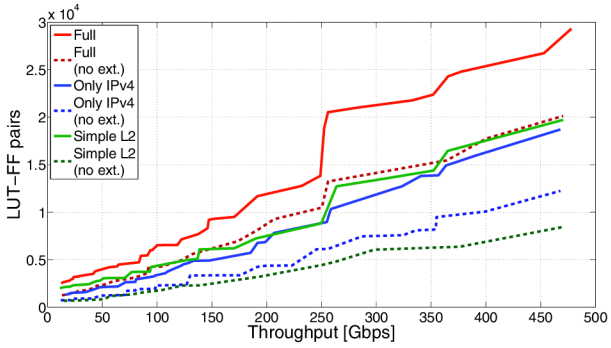


Fig. 9. Comparison of the FPGA resource utilization versus throughput Pareto sets for the tested protocol stacks.

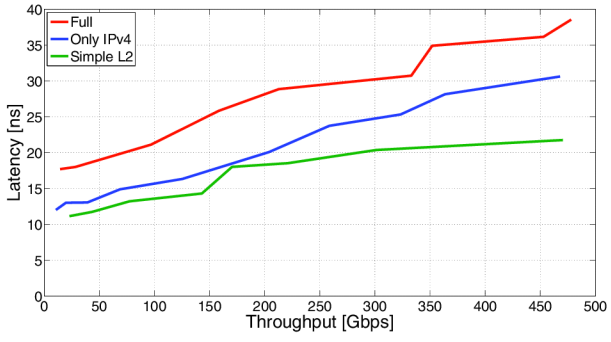


Fig. 10. Comparison of the latency versus throughput Pareto sets for the tested protocol stacks.

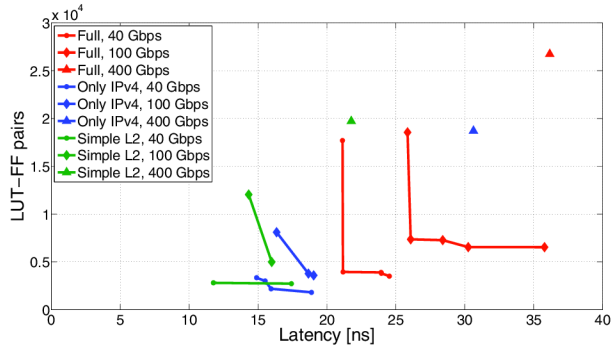


Fig. 11. Pareto sets for three given protocol sets and three target throughputs.

A careful design space exploration is very important for our parser. For example, the parser of the full protocol stack optimized for the latency uses 17 685 LUT-FlipFlop pairs to achieve near 100 Gb/s throughput with the latency of only 21.1 ns (see Tab. I), while the parser optimized for resources uses only 6 536 LUT-FlipFlop pairs to achieve the throughput just over 100 Gb/s, but with the latency of 35.8 ns (see Fig. 11).

Finally, Fig. 12 illustrates the complete Pareto set of solutions in the (latency, throughput, area) space for the full protocol stack. To create the 3D surface in the figure, the bottom (latency, throughput) plane was divided into rectangles of sizes (1 ns $\times$ 10 Gb/s) and the smallest solution that satisfies the required latency and throughput was found for each rectangle. Therefore, each horizontal level of the surface represents one solution from the Pareto set. Finer-grained division of the (latency, throughput) plane would result in more solutions, but also in less readable image.

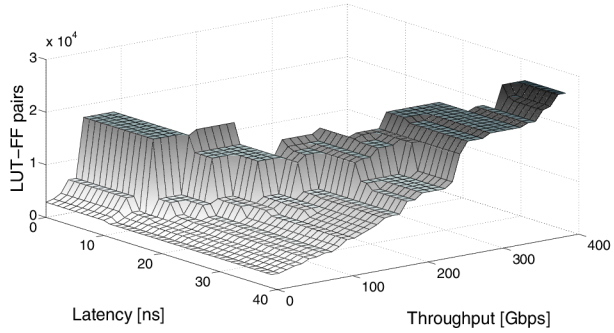


Fig. 12. 3D surface plot of the Pareto set

## VI. Conclusion

This paper introduces a highly configurable packet parser for FPGA, which achieves throughput in the range of tens to hundreds of Gb/s and is usable in a variety of applications. The key concept is a selective pipelining, which allows to find the best fitting solution with regards to the requirements. The parser uses only 1.19 % of the Virtex-7 870HT FPGA resources to achieve a throughput over 100 Gb/s and 4.88 % for a throughput over 400 Gb/s, which leaves most of the FPGA resources free for implementing other functions of target applications.

This work also presents the methodology of a modular parser design and demonstrates the need for a thorough exploration of the solution space. Moreover, it suggests several capabilities that a High-Level Synthesis system should possess to succeed in area of packet parsers creation.

## Acknowledgement

This research has been supported by the “CESNET Large Infrastructure” project no. LM2010005 funded by the Ministry of Education, Youth and Sports of the Czech Republic, the “DMON100” project no. TA03010561 funded by the Technology Agency of the Czech Republic, BUT project FIT-S-14-2297 and the IT4Innovations Centre of Excellence CZ.1.05/1.1.00/02.0070.

## References

[1] 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*, oct. 2011, pp. 12–23.

[2] F. Braun, J. Lockwood, and M. Waldvogel, “Protocol wrappers for layered network packet processing in reconfigurable hardware,” *Micro, IEEE*, vol. 22, no. 1, pp. 66–74, 2002.

[3] P. Kobierský, J. Kořenek, and L. Polčák, “Packet header analysis and field extraction for multigigabit networks,” in *Proceedings of the 2009 12th International Symposium on Design and Diagnostics of Electronic Circuits&Systems*, ser. DDECS. Washington, USA: IEEE Computer Society, 2009, pp. 96–101.

[4] 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*, aug. 2007, pp. 648–651.

[5] “Xilinx microblaze soft processor,” Xilinx, Inc., http://www.xilinx.com/tools/microblaze.htm.

[6] “Xilinx virtex–7 fpga family,” Xilinx, Inc., http://www.xilinx.com/products/silicon-devices/fpga/virtex-7.

[7] C. Kozanitis, J. Huber, S. Singh, and G. Varghese, “Leaping multiple headers in a single bound: Wire-speed parsing using the kangaroo system,” in *IEEE INFOCOM*, mar. 2010.

## A.5 Paper 5

**Fast Lookup for Dynamic Packet Filtering in FPGA**

# Fast Lookup for Dynamic Packet Filtering in FPGA

Lukáš Kekely, Martin Žádník, Jiří Matoušek, Jan Kořenek
IT4Innovations Centre of Excellence
Brno University of Technology, Czech Republic
Email: ikekely,izadnik,imatousek,korenek@fit.vutbr.cz

***Abstract*—Rapidly growing speed and complexity of computer networks impose new requirements on fast lookup structures which are utilized in many networking applications (SDN, firewalls, NATs, etc.). We propose a novel lookup concept based on the well-known cuckoo hashing, which can achieve good memory utilization, supplemented by a binary search tree for offloading the colliding keys and supporting LPM lookup. We also propose a hardware architecture implementing this lookup concept in the FPGA. Our solution is suitable for lookup of the variable-length keys in 100+ Gbps networks. Memory utilization of the proposed concept is thoroughly evaluated and it is shown that the concept is scalable to external memory components.**

***Keywords*—*Cuckoo hash; binary search; packet filtering; FPGA***

## I. Introduction

Field Programmable Gate Arrays (FPGA) are popular platforms utilized in networking applications targeting high-speed packet processing (e.g. [1]). We propose a fast lookup concept designed specifically for FPGA-oriented platforms. The concept combines two well-known memory-oriented lookup algorithms – cuckoo hashing [2] and binary search tree (adapted for best/longest prefix matching [3]). Each algorithm efficiently complements the other in area where the other fails. The concept achieves almost 100% memory utilization with efficient utilization of the memory and logic resources in comparison to the TCAM or Hash-CAM concepts [4]. At the same time, our concept allows fast lookups (200 mil. lookups/s designed for 100+ Gbps solutions). Our contributions also include: (a) the possibility to utilize external memory when the number of rules cannot fit in the internal FPGA memory, (b) increasing the lookup functionality with the longest prefix match, (c) efficient implementation of the whole scheme in FPGA including the update logic enabling on-the-fly updates and (d) evaluation of the concept in terms of achievable resources utilization.

## II. Related Work

The goal of cuckoo hashing [2] is to reduce the number of memory accesses during a lookup and thus speeding-up a lookup operation. Standard cuckoo hashing utilizes two hash tables with two different hash functions but it can be generalized for a higher number of hash tables/functions. There has been an implementation of cuckoo hashing in FPGA for the purpose of pattern matching [4]. This architecture contains dedicated matching blocks for all patterns of the same length (up to the length of 16 characters). Each matching block consists of two cuckoo hash tables for storing addresses to the database of patterns. The architecture also contains multiplexers and a control logic which together allow performing either a pattern matching operation or a pattern database update (pattern insertion or deletion). The approach offers only medium memory utilization since it does not utilize any type of overflow memory and also cannot scale well to external memory since it is tailored to the internal FPGA memory.

The advanced lookup procedures also include prefix matching (PM, i.e. there is a single prefix for a given key in the set but it is not known apriori) and longest prefix match (LPM, i.e. selecting the longest matching prefix from the set for a given key). Although the LPM itself is out of the primary scope in this paper, the unique combination of cuckoo hash and binary search tree renders it possible for our implementation to support LPM lookup.

## III. Design and architecture

The core functionality of our lookup schema is based on cuckoo hashing principle due to a very fast lookup with only a few memory accesses needed for each search. This feature favors the usage of cuckoo hashing even on architectures with limited memory interface throughput (e.g. external memory). On the other hand, cuckoo hashing can suffer from a low achievable utilization of the memory caused by hash conflicts. To address this problem, our design augments basic cuckoo hashing principle by the usage of a stash for offloading the conflicting keys. The proposed design of cuckoo hashing with the stash is not entirely new. It has been already described in [5], where the authors proposed and evaluated the usage of only a very small stash (capacity under 5 keys implemented in TCAM) to improve the worst case memory utilization of cuckoo hashing.

In our design we propose and evaluate the usage of a significantly larger stash – a stash with the capacity comparable to the capacity of the used cuckoo hash tables to improve not only the worst case but also to improve average memory utilization. Furthermore, our stash also supports LPM lookups, thus augmenting the lookup functionality of the basic cuckoo hashing. The lookup support of not only the whole keys but also key prefixes can be very useful in many different areas (e.g. packet filtering). Instead of TCAM, we propose an FPGA implementation of a well-known binary search algorithm adapted for the LPM lookup (described in [3]) as an effective approach to implement the larger stash.

The binary search offers basically the opposite features in comparison with the cuckoo hashing – the key lookup requires relatively large number of subsequent memory accesses, but the achievable memory utilization is always 100 %. Because of the large number of memory accesses, binary search based lookup should be implemented only in the internal FPGA memory. In order to achieve high lookup throughput, the

978-1-4799-4558-0/14/$31.00 ©2014 IEEE

implementation of the binary search must not be sequential but rather divided into pipelined stages. This can be achieved by establishing a tree structure in the searched array (binary search tree – BST) and slicing it by the tree levels (each tree level forms a pipeline stage). Finally, the functionality of update operations in the described BST can be easily implemented in the hardware with support of on-the-fly updates.

### A. Lookup engine interface and functionality

We start the description by the general design of an interface and functionality of a virtual lookup engine (either cuckoo or BST). Both engines implement the same interface independently on the details of their lookup procedure. The signals can be divided into 3 basic groups: input, output and configuration. The only input of a lookup engine is the value of a key to search. The lookup implementation should be able to process new input key every clock cycle. For each input key, the engine produces one result on the output based on performed lookup. The lookup result consists of arbitrary data (e.g. routing decision, matched key identification) associated with the searched key and one bit information about the key lookup success (Found). When the input key is not found, the value of data on the output is unspecified (invalid).

The lookup engine (and its interface) is configurable by these three basic generic parameters: **key width** (maximum width of key representation in bits), **data width** (width of data representation in bits), **maximum capacity** (theoretical size limit for the set of keys, representation may differ).

### B. Cuckoo hash lookup engine

Fig. 1 depicts a basic schema of cuckoo hash engine implementation. The lookup process starts by parallel computing of key hash values (outputs of hash blocks). As the basis for the hash blocks we utilize CRC implementation generated for commonly used polynomials. The lookup continues with hash values being used as addresses for reading records from hash tables in memory. Each record forms a pair composed of a key and data associated with the key. A record can also be stored in a register outside the tables (the purpose of the register is explained in the next paragraph). Subsequently, the input key is compared with the keys from the memory (and the register) records for equality. At most one comparison may be successful, because each unique key appears only in a single place at a time. Therefore, aggregation of result is very simple – if none of the compared keys is equal to the searched key, the found flag is not set, otherwise it is set and data associated with the matching key are provided.

Update of an active key set is entirely managed by the reconfiguration controller based on requests received from the configuration interface. When inserting a new key, the controller can take advantage of the reconfiguration register included in the lookup path. Using this register the controller can evict records from hash tables on-the-fly preserving the set of active keys. More precisely, the insertion of a new key $x$ starts with storing $x$ in the register. Then all possible locations for $x$ in the hash tables are checked sequentially. If one of them is empty, $x$ is inserted into the table and the reconfiguration ends. Otherwise a victim $y$ is selected and evicted from the table, leaving free space for $x$. The evicted record $y$ is actually

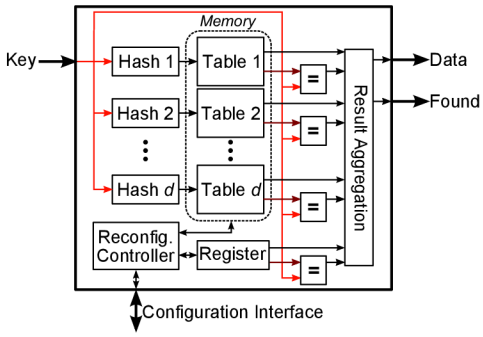


Fig. 1. Conceptual schema of cuckoo hash based lookup engine.

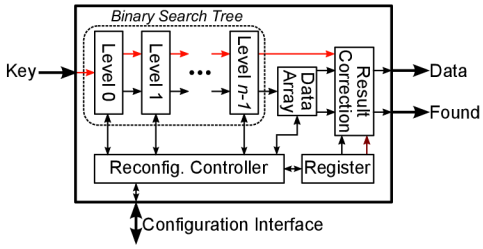


Fig. 2. Conceptual schema of binary search tree based lookup engine.

swapped with $x$ and the insertion continues with $y$ except $x$ cannot be selected as the next victim. The reconfiguration cycle can repeat itself multiple times, until the register is freed or can even repeat itself infinite times when a chain of collisions occurs. Until the register is freed the cuckoo hash engine is considered full. Deletion of a key is possible even during active insertion reconfiguration. Deletion of $x$ starts by pausing the reconfiguration process and continues with sequential checking of all possible locations for $x$ (i.e. the register and a single position in each table). If a key identical to $x$ is found in one of those positions, it is invalidated. After the deletion ends, the reconfiguration process is resumed.

The maximum capacity of the cuckoo hash engine can be configured by two values: $d$ – the number of used hash tables (hash functions) and $t$ – the size of individual table. Theoretical capacity limit is defined by formula $C_{cuckoo} = d \times t + 1$. The plus one accounts for the additional reconfiguration register.

### C. Binary search tree lookup engine

Fig. 2 depicts a basic schema of our BST lookup engine. The engine starts the lookup by a pipelined and sequential search of an input key (red arrows) through the levels of the tree. Each tree level forms a pipeline stage with its dedicated piece of memory and a key comparator. The output of a stage is an address of a node where to continue binary search in the next tree level and the searched key. The address from the last tree level is used to address the data array containing associated data to the key. The lookup result must be corrected according to a record in the reconfiguration register due to atomicity of operations.

Update of an active key set is entirely managed by the reconfiguration controller based on requests received from the configuration interface. The controller can take advantage of

the single reconfiguration register included in the lookup path during the update. More precisely, the update (deletion or insertion of $x$) starts with storing the record $x$ in the register. Subsequently, the update process consists of three sequential steps. (1) The key $x$ is searched in the tree sequentially. The search must fail when inserting $x$. The search must succeed when deleting $x$. (2) The record $x$ is activated in the register to correct the lookup process in the last stage. (3) Sequential reconfiguration is performed to merge $x$ into the nodes and the data array. Finally, the update process ends and the reconfiguration register is freed. Deletion and insertion share resources and cannot be active together as in cuckoo hash engine. The engine can become full only after successful insertion and can become empty again only after successful deletion.

The capacity of the BST based engine can be configured by the number of BST levels $l$. The capacity is then defined by formula $C_{bst} = 2^l - 1$ when adaptation for LPM is not used or $C_{bst} = 2^{l-1} - 1$ when LPM lookup is supported. Our implementation supports the adaptation for LPM, but if LPM is not needed, it can be easily modified (simplified) to support only precise key lookup gaining two times higher capacity.

### D. Top-level lookup engine

Top-level engine instantiates both Cuckoo and BST engine in parallel. The lookup of an input key is also performed in parallel in both engines. The results are then stored in FIFOs, because the two engines do not have same processing delays. Result aggregation then selects data from engine with successful lookup. When both engines successfully find a key, the result from cuckoo hash is preferred, because in that case the result from BST is only for a matching prefix, but the result from cuckoo hash is for the whole matching key.

Reconfiguration of the key sets in both engines is managed by the top level reconfiguration controller. All updates for prefixes are directly forwarded into the BST stash. Deletions of the whole keys are implemented in both engines in parallel. Insertions of the whole keys are forwarded into the cuckoo hash. If cuckoo hash is full (its reconfiguration register is occupied) and new key insertion is requested, then the key that is currently in cuckoo hash reconfiguration register is moved into the stash and the new key is inserted into cuckoo hash. The top-level engine is full when both the cuckoo hash and the BST stash are full. Furthermore, in our implementation the configuration interface of the top-level lookup engine is connected to the block with address decoder and registers for key, data, requests and status flags. This block is then accessible from the software using standard memory interface. This way the management of the active key set can be easily controlled from the software.

The maximum capacity of the cuckoo hash with stash lookup engine can be defined by three parameters: parameters $d$ and $t$ of the cuckoo hash and the stash size $s$. Theoretical capacity limit is then defined by formula $C_{total} = d \times t + 1 + s$.

## IV. Evaluation and results

The proposed architecture was implemented in VHDL and synthesized into FPGA. We conducted experiments to evaluate achievable memory utilization and FPGA resources consumption in different configurations of the architecture. The results of these evaluations are summed up in this section.

We start the evaluation by experiments on achievable memory utilization of our concept. The achieved utilization can be computed in two basic ways: $U_{cuckoo} = (n-m)/C_{cuckoo}$, $U_{total} = n/C_{total}$, where $n$ is the total number of successfully inserted keys before the memory became full and $m$ is the number of keys that resides in the stash. Because, our implementation uses stash which can be always filled up to 100 % of its capacity, we can always put $m = s$. The values of $n$ must be acquired from the test runs.

In the first series of tests we have evaluated the relation between achievable memory utilization of cuckoo hash and the used sizes of stash for different parameters. The results of these evaluations are shown in the graphs in Fig. 3 and 4. We have tested three different values of $d$ parameter (2, 3 and 4 not depicted in the figures), three different values of $t$ parameter (128, 1 024 and 8 192) and multiple values of $s$ (from 0 to $t$). We have also tested different key sizes (32 b, 64 b and 128 b), but the achieved results have been very similar, therefore we do not show different graphs for each key size. The memory utilization plotted in the graphs is $U_{cuckoo}$ and the size of the stash ($s$) is plotted as a portion of $t$. The graphs show mean (thick darker lines) and minimal resp. maximal (thin lighter lines) achieved utilizations from 10 000 tests with random generated keys for each combination of values of $d$, $t$ and $s$. From data plotted in the graphs it is clear that the mean achieved memory utilization of cuckoo hash is independent on the values of $t$. Parameter $t$ only influences the difference between minimal and maximal achieved utilization, when the span is higher for smaller values of $t$.

Moreover, Fig. 3 shows that the influence of stash size on the achievable memory utilization is significant for two cuckoo hash tables – the mean utilization raises from 50 % in the case without the stash to 75 % with $s = t/10$ or even around 90 % for $s > t/2$. Also the differences between minimal and maximal achieved utilizations are reduced with the raising size of stash. Fig. 4 shows that the importance of stash in case of more than two cuckoo hash tables is not that high as for two tables. But it contributes in achieving nearly 100 % mean memory utilization of cuckoo hash tables.

The second series of memory utilization tests is oriented on a thorough examination of achievable memory utilizations for a few selected sizes of stash. The results of these evaluations are shown in the graphs in Fig. 5 and 6. Here we have also tested three different values of $d$ parameter (2, 3 and 4 not depicted), but only a single value of $t = 1\,024$ and only a few values of $s$ (0, $t/64$, $t/16$, $t/4$, $t/2$ and $t$). The graphs show histograms of probability (percentage of all conducted tests) that achieved precisely the specified utilization ($U_{cuckoo}$ used) with highlighted mean (dashed line) and minimal resp. maximal utilizations (points). The area under each histogram line is exactly 100 % even though the individual values are rather small. The results are from 1 000 000 tests with random generated keys for each combination of values of $d$ and $s$. From data plotted in the graphs it is clear that the dispersion of achieved utilizations is lower for the rising stash size. Also the effect of stashes with size $s < t/16$ for the cuckoo hash with $d > 2$ is negligible.

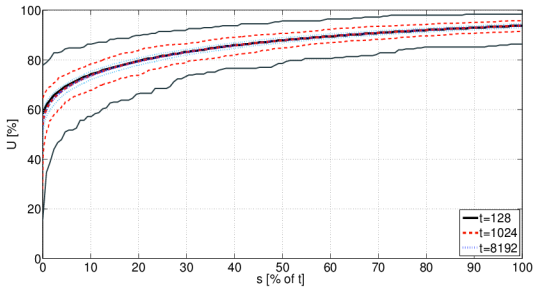


Fig. 3. Achievable memory utilization for cuckoo hash with two tables ($d = 2$) for different sizes of stash.

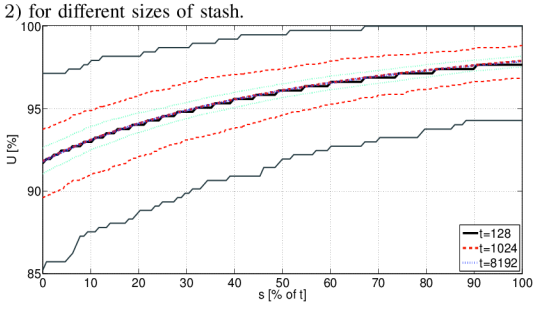


Fig. 4. Achievable memory utilization for cuckoo hash with three tables ($d = 3$) for different sizes of stash.

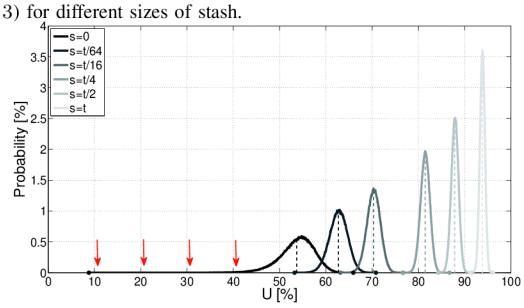


Fig. 5. Probability distribution of achievable memory utilization for cuckoo hash with two tables ($d = 2$, $t = 1\,024$).

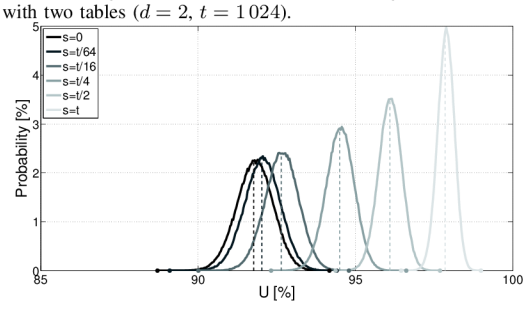


Fig. 6. Probability distribution of achievable memory utilization for cuckoo hash with three tables ($d = 3$, $t = 1\,024$).

The dispersion reduction is noticeable especially for the cuckoo hash with two tables (Fig. 5). For two tables without a stash there is a very real chance of achieving memory utilization that is significantly lower than the mean utilization (marked by red arrows). The solution to this problem is even a relatively small stash ($s = t/64$ or $s = t/16$). This particular situation is very important when cuckoo hash is implemented using large external memory to store cuckoo hash tables. The bottleneck in such an implementation lays in the throughput of external memory interface, which limits the number of usable cuckoo hash tables usually to only 2. These results suggests that stash of size $s = t/64$ or $s = t/16$ can significantly improve the achievable memory utilizations in exactly this case. So for example, the implementation of cuckoo hash with $d = 2$ and $t = 2^{20}$ in external memory require stash with size only $s = 2^{20}/16 = 65\,536$ to achieve mean external memory utilization of 70 % (mean capacity over 1.5 million keys) with very low chance to achieve utilization under 65 %.

TABLE I. FPGA RESOURCES REQUIREMENTS AND MEMORY UTILIZATIONS OF OUR LOOKUP ENGINE IMPLEMENTATION.

| Key Width | d | t | s | FPGA Resources LUTs | FFs | BRAMs | Frequency [MHz] | Mean Utilization | Mean Capacity |
|---|---|---|---|---|---|---|---|---|---|
| 32 | 2 | 8 192 | 2 047 | 3 721 | 2 111 | 45 | 264.116 | **83.5 %** | **15 388** |
| 32 | 3 | 8 192 | 4 095 | 4 138 | 2 221 | 71 | 265.437 | **96.7 %** | **27 711** |
| 128 | 2 | 1 024 | 255 | 8 336 | 4 059 | 15 | 257.631 | **83.5 %** | **1 923** |
| 128 | 3 | 1 024 | 511 | 9 564 | 4 304 | 23 | 263.704 | **96.7 %** | **3 463** |

Finally, we present the FPGA resources requirements of our top-level lookup engine in selected configurations. The requirements in terms of LUTs, registers and BlockRAMs are given in Tab. I together with the achievable clock frequencies. Values in tables are acquired from the synthesis by XST tool for the XilinxVirtex-7 870HT FPGA and data width of 32 bits. Variable key widths (32 and 128 bits as lengths of IPv4 and IPv6 addresses were selected) and capacity parameters $d, t, s$ are given in the table. Tab. I also presents mean achievable memory utilization ($U_{total}$) and capacity based on test results presented earlier in this section. The achieved frequencies over 200 MHz and the fact that each lookup implementation is capable of one lookup on each clock cycle suggest, that our architecture is capable of over 200 million lookups per second, which is sufficient for packet filtering on 100+ Gbps networks.

## V. CONCLUSION

The paper proposed a viable concept for fast packet filtering for FPGA. The proposed architecture leverages the combination of the cuckoo hash engine with BST engine with a focus on parallel implementation in FPGA. The results of evaluation show that the concept allows not only fast lookups for every arriving packet on the 100+ Gbps links but also effective utilization of FPGA resources.

## ACKNOWLEDGEMENT

This research has been supported by the grant MVCR VG20102015022, BUT project FIT-S-14-2297 and the IT4-Innovations Centre of Excellence CZ.1.05/1.1.00/02.0070.

## REFERENCES

[1] J. Naous and et al., "Implementing an openflow switch on the netfpga platform," in *Proceedings of ANCS*, NY, USA, 2008, pp. 1–9.

[2] R. Pagh and F. F. Rodler, "Cuckoo hashing," *J. Algorithms*, vol. 51, no. 2, pp. 122–144, May 2004.

[3] B. Lampson, V. Srinivasan, and G. Varghese, "Ip lookups using multiway and multicolumn search," in *INFOCOM*, 1998, pp. 1248–1256.

[4] T. Tran and S. Kittitornkun, "Fpga-based cuckoo hashing for pattern matching in nids/nips," in *MNGNS*, ser. LNCS, 2007.

[5] A. Kirsch, M. Mitzenmacher, and U. Wieder, "More robust hashing: Cuckoo hashing with a stash," in *ESA*, ser. LNCS. Springer, 2008.

## A.6 Paper 6

**Software Defined Monitoring of Application Protocols**

# Software Defined Monitoring of Application Protocols

Lukáš Kekely, Jan Kučera, Viktor Puš, Jan Kořenek, and Athanasios V. Vasilakos

**Abstract**—With the ongoing shift of network services to the application layer also the monitoring systems focus more on the data from the application layer. The increasing speed of the network links, together with the increased complexity of application protocol processing, require a new way of hardware acceleration. We propose a new concept of hardware acceleration for flexible flow-based application level traffic monitoring which we call Software Defined Monitoring. Application layer processing is performed by monitoring tasks implemented in the software in conjunction with a configurable hardware accelerator. The accelerator is a high-speed application-specific processor tailored to stateful flow processing. The software monitoring tasks control the level of detail retained by the hardware for each flow in such a way that the usable information is always retained, while the remaining data is processed by simpler methods. Flexibility of the concept is provided by a plugin-based design of both hardware and software, which ensures adaptability in the evolving world of network monitoring. Our high-speed implementation using FPGA acceleration board in a commodity server is able to perform a 100 Gb/s flow traffic measurement augmented by a selected application-level protocol analysis.

**Index Terms**—Network monitoring, acceleration, security, FPGA, L7

## 1 Introduction

Modern network engineering and security heavily rely on the network traffic monitoring. The requirements imposed on the quality of network security monitoring information often lead to the requirement to process unsampled network traffic. That ability is crucial in order to detect even single-packet attacks. A golden standard in the area of network monitoring is a flow measurement. A monitoring device collects basic statistics about the network flows at the Internet and Transport layers and reports them to a central storage collector using a handover protocol such as NetFlow [1] or IPFIX [2]. Flow measurement is a stateful process, because for each packet the flow state record is updated in the device (e.g. packet counters are incremented), and only the resulting numbers are exported. This also implies that some information is lost in the monitoring process and that the flow collector (where further data processing is usually done) has a limited view on the network. While a number of researchers focus on harvesting knowledge from the existing flow data, we argue that the ability to analyze *application layer* in the monitoring process is crucial for improvement of the quality and flexibility of network monitoring. This is illustrated by the recent infamous Heartbleed bug in the SSL implementation. While it is impractical (if not impossible at all) to detect the Heartbleed attack by analyzing the transport layer flow data, its detection at the application layer is trivial.

The ongoing trend in the field of application layer monitoring is towards creation of richer flow records [3], [4], [5], carrying some extra information in addition to the basic flow size and timing statistics. The added information often include values from the application layer protocol headers, such as HTTP, DNS etc. It seems that the ability to analyze application layer in the monitoring process is crucial for improvement of the quality of network threat detection, because more and more of the network functionality is being shifted up in the protocol stack.

Implementation of the application level flow monitoring with a commodity CPU is certainly possible, yet its throughput is limited mainly by the performance of the processors [6]. It should be noted that every newly arrived packet is inevitably a cache miss in the CPU. On the other hand, ASICs and FPGAs offer much better possibilities in terms of throughput. However, a fixed solely hardware implementation may face the flexibility issues, since the evolving nature of network threats implies the need for fast changes of the monitoring process, quickly making fixed hardware devices obsolete. Many papers proposing high-speed hardware architectures for the most timing-critical operations necessary in flow monitoring were published. Those operations include packet header parsing, packet classification, counters management and pattern matching. However, most of the proposed architectures have never been practically deployed. We conceive that this is because the effort is usually spent only on the improvement of the performance features, but flexibility, ease of use and speed of response to newly emerged problems are neglected.

The aim of this paper is to (1) strike a balance between the system throughput and flexibility/programmability and to (2) offer a configurable trade-off to the above, but

- *L. Kekely, J. Kučera and V. Puš are with CESNET a. l. e., Zikova 4, 160 00 Prague, Czech Republic. E-mail: {kekely, jan.kucera, pus}@cesnet.cz.*
- *J. Kořenek is with the IT4Innovations Centre of Excellence, Faculty of Information Technology, Brno University of Technology, Božetěchova 2, 612 66 Brno, Czech Republic. E-mail: korenek@fit.vutbr.cz.*
- *A. V. Vasilakos is with the Department of Computer Science, Electrical and Space Engineering, Lulea University of Technology, Sweden. E-mail: athanasios.vasilakos@ltu.se.*

*Manuscript received 17 June 2014; revised 20 Feb. 2015; accepted 6 Apr. 2015. Date of publication 15 Apr. 2015; date of current version 15 Jan. 2016. Recommended for acceptance by Y.-D. Lin.*
*For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.*
*Digital Object Identifier no. 10.1109/TC.2015.2423668*

0018-9340 © 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

mainly to (3) endorse a progressive adoption of network monitoring subtasks to the hardware accelerator, motivated solely by the real needs of the networking community.

Our key idea is that even the advanced application-layer processing task usually need to observe only some network flows, representing only a small fraction of traffic. An example can be a DNS analyzer, since DNS traffic typically represents no more than 1 percent of all network packets. Other applications may utilize even better offload, since they need to observe only a small amount of packets within each flow for their full functionality. Let a HTTP header analyzer be an example, since the HTTP header is typically located in the first few packets of the network flow. Please note that our method never discards packets that are relevant for the particular monitoring application.

We only offload the processing of bulk traffic that is not (or no longer) interesting for the application-layer processing tasks into the hardware accelerator. The offload of measurement is controlled on a per flow basis by the monitoring software and adjusted in real time to its current needs. Offload control is realized through unified interface by dynamically specifying a set of rules. These rules are installed into the accelerator to determine the type of packet offload (=preprocessing acceleration) used for individual network flows. The preprocessing method that best aids the performance and does not decrease the required precision of advanced software processing is selected. Due to the unified control interface the proposed system is very flexible and can be used for a wide range of network monitoring applications.

Furthermore, the whole system is designed to be easily extensible at two different levels. At the software side, monitoring plugins can be easily added to the system. This brings the possibility of rapid development and deployment of new monitoring applications, for example as a reaction to a new network security threat. Once the functionality of software task is verified and stable enough, the second level of system extensibility can be employed to further speed-up the task. Various packet processing and data aggregation routines can be relocated directly into the hardware accelerator.

The paper is structured as follows: The following section provides analysis of real-life network traffic from the application monitoring point of view. In Section 3 we make use of the analysis outcomes to design the concept of hardware accelerator tightly coupled to monitoring software applications. Section 4 provides experimental results of our work. Section 5 presents notable related work and Section 6 concludes our paper.

## 2 Analysis

We start the paper with an analysis of traffic properties in a real high-speed backbone network. Based on the measured characteristics we then optimize the design of our SDM system to achieve optimal performance when deployed in real networks. All of the measurements in this paper were conducted in the high-speed CESNET backbone network. CESNET is Czech National Research and Educational Network which has optical links operating at speeds up to 100 Gbps and routes mainly IP traffic. It serves around 200,000 users. We conduct all of our measurements during the standard working hours. To get a basic view of the network traffic character, we measure mean size of packets in bytes, mean size of flows in packets and mean time duration of flows. Because we aim for the application protocols, we measure these characteristics not only for the whole network traffic on the link, but also for the selected applications. We select some of the most commonly used application protocols and services such as HTTP, HTTPS, DNS, email (SMTP, POP3 and IMAP), SSH, RTMP, FTP and others. Furthermore, we measure the percentage of these protocols in the captured traffic in terms of flows, packets and bytes.

TABLE 1
Basic Statistical Characteristics of Network Data Grouped by the Service

| | Traffic portion in | | | Average | | |
|---|---|---|---|---|---|---|
| | flows [%] | packets [%] | bytes [%] | flow size [packets] | flow time [s] | pckt size [Bytes] |
| HTTP | 26.62 | 48.33 | 51.81 | 59.2 | 7.137 | 983.0 |
| HTTPS | 18.18 | 31.12 | 29.75 | 51.3 | 8.591 | 816.7 |
| SSH | 2.66 | 1.42 | 1.09 | 11.7 | 17.167 | 241.2 |
| RTMP | 0.02 | 1.01 | 1.24 | 2,066.8 | 57.432 | 1,001.2 |
| DNS | 24.10 | 0.79 | 0.19 | 1.1 | 0.153 | 205.9 |
| Email | 1.00 | 0.72 | 0.56 | 16.8 | 2.957 | 581.6 |
| ICMP | 1.91 | 0.60 | 0.50 | 1.9 | 3.206 | 91.3 |
| RDP | 3.37 | 0.53 | 0.31 | 4.7 | 2.731 | 468.4 |
| NTP | 1.53 | 0.41 | 0.21 | 8.8 | 4.142 | 359.5 |
| FTP | 0.38 | 0.01 | 0.01 | 2.3 | 1.234 | 75.8 |
| SIP | 0.00 | 0.00 | 0.00 | 5.0 | 23.611 | 421.1 |
| others | 20.23 | 15.06 | 14.33 | 27.2 | 7.536 | 839.6 |
| all | | | | 32.0 | 6.432 | 872.2 |

Table 1 shows the results of the basic network traffic analysis. The table shows that the observed statistics differ greatly depending on the specific service. The largest portion of network traffic is conveyed by the HTTP protocol which accounts for more than a quarter of all flows and around half of all packets and bytes. Moreover we can see that HTTP flows and packets are generally larger (heavier) in number of packets and bytes and longer in time than average. Another large amount of total traffic belongs to HTTPS, which has very similar observed characteristics as HTTP. These two protocols (HTTP and HTTPS) together cover majority of all network traffic—nearly a half of all flows and around four fifths of the data. Therefore, the possibility of their further analysis is certainly desirable. A large amount of flows also belong to the DNS protocol (nearly one quarter), but this number is highly disproportional to the DNS total packet and bytes percentage. DNS flows are generally very small (light) with majority of them consisting of only one small packet. Also ICMP, which covers majority of non-TCP/non-UDP flows on the network, has similar character of flows as DNS with very small and short flows. The opposite type of disproportional flows and packets percentages as DNS and ICMP has RTMP protocol (Flash multimedia streaming), which covers only a tiny portion of flows, but they are all extremely heavy and long.

The *distribution* of packet lengths is another interesting characteristic of the network. The majority of packets are either very long (over 1.300 B: 57 percent) or very short (under 100 B: 35 percent). Especially dominant are both extremes from the range of lengths supported by the

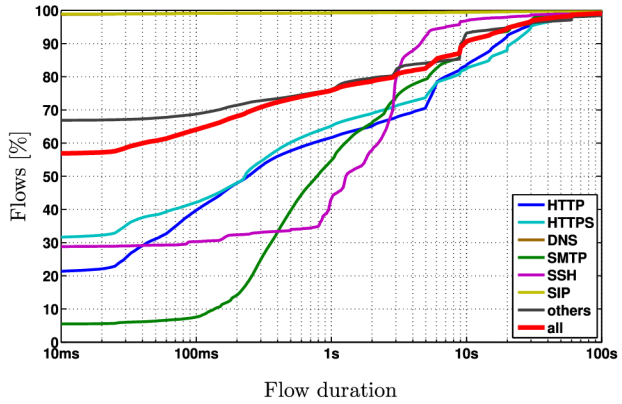


Fig. 1. Cummulative distribution functions of flow durations.

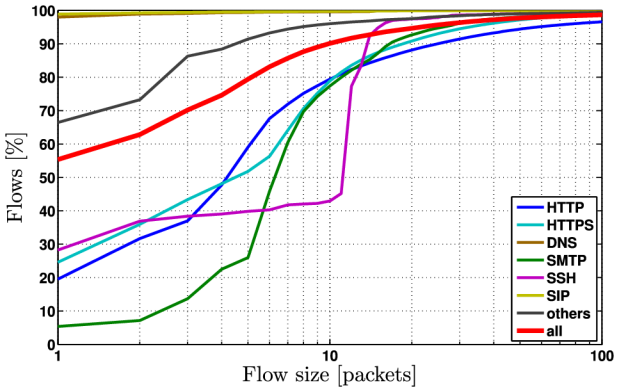


Fig. 3. Cummulative distribution functions of flow sizes.

Ethernet standard—42 and 1,500 B. Medium sized packets are not very common.

In Table 1 we have already shown basic information about mean flow durations. Further information about the flow time durations for the selected application protocols can be seen in Fig. 1. Each line in the graph shows the percentage of flows that last shorter than the given duration. Generally (red thick line) over $\frac{2}{3}$ of all flows are shorter than 100 ms and only a tenth of them exceed 10 s. Also majority of DNS and SIP flows have a duration under 10 ms.

While Fig. 1 shows further information about flow duration, it does not say anything about time distribution of packets inside the flows. Weights of individual flows are also not considered. A better look at packet timing inside the flows can be shown by measuring the relative arrival times of packets from the start of the flow. Thus, the first packet of each flow has the zero relative arrival time and its absolute arrival time marks the starting time of that flow. Then, each subsequent packet has a relative arrival time equal to the difference of its absolute arrival time and the marked start of the flow. Results of this measurement are shown in Fig. 2. The graph shows that on average (red thick line) only a small portion of all packets arrive right after the start of the flow—;only a fifth of all packets arrive during the first second of the flow. This fact leads to the conclusion that flows with short duration carry only a very few packets. The conclusion is further strengthened by the fact that the majority of flows have a very short duration.

Table 1 contains the information about mean flow sizes for selected application protocols and services. Further information about flow sizes can be seen in Fig. 3. Each line of the graph shows the percentage of flows that consists of fewer packets than a given number. On average (red thick line) only a tenth of all network flows have more than 10 packets. Also, virtually all DNS and SIP flows consist of a single packet.

Fig. 3 does not clearly say anything about the percentage of all packets carried by flows of different sizes. It is known that high-speed network traffic has a heavy-tailed character of flow size distribution [7], [8]. The heavy-tailed character of flow size distribution derived from the measured values is shown in Fig. 4. The graph shows the portions of all packets carried by the specified percentage of the heaviest flows on the network. It can be seen that on average (red thick line) 0.1 percent of the heaviest flows carry around 60 percent of all packets and 1 percent carry even around 85 percent. An exception to the heavy-tailed distribution of flow sizes is the DNS protocol. On the other hand, SIP and SSH protocols have a heavier tail than average.

Our work relies on the following consequence of the heavy-tailed character of network traffic: by selecting a small percentage of the heaviest flows, we can cover the majority of packets. The problem then lies in an effective prediction of which flows are the heaviest. More accurately, it lies in a capability to recognize the heaviest flows only from the properties of their first few packets. The simplest method of

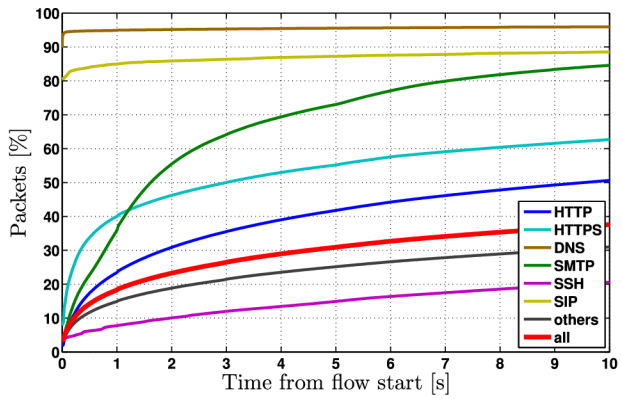


Fig. 2. Cummulative distribution functions of packet arrival times.

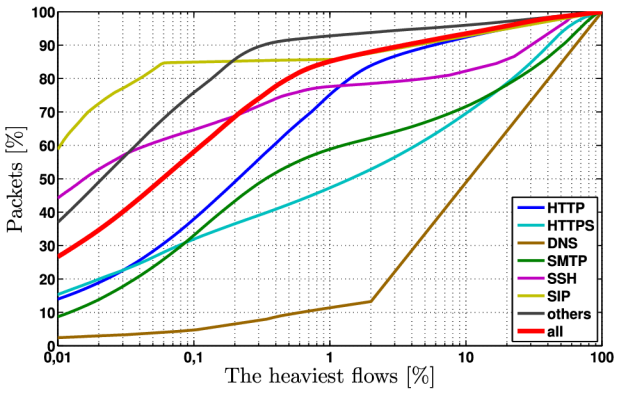


Fig. 4. Portions of packets carried by the percentage of the heaviest flows.

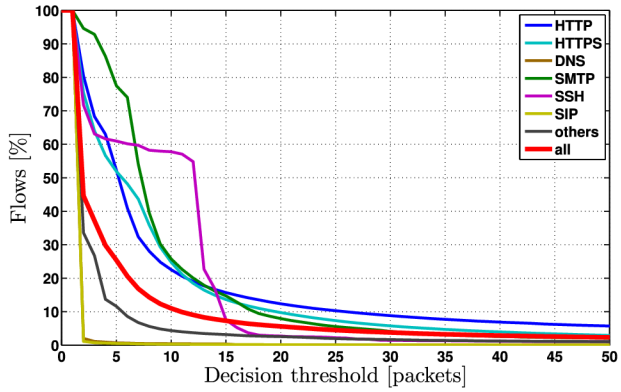


Fig. 5. Heavy flow detection using the simple method—portions of selected flows.

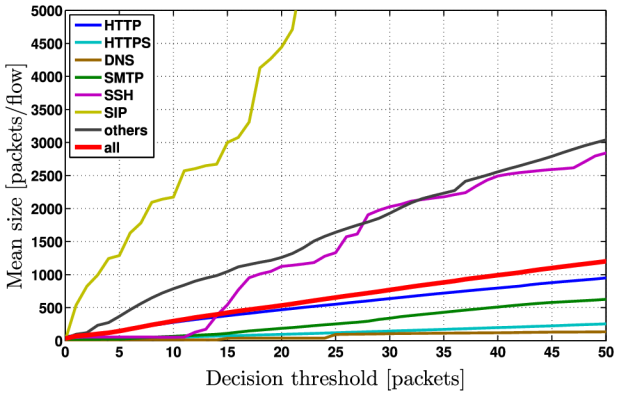


Fig. 7. Mean number of captured packets per flow in flows selected using the simple method.

this recognition is based on a rule that every flow is considered heavy after arrival of its first $k$ packets for some selected decision threshold $k$. The main advantage of this method is just its simplicity—no additional packet analysis nor advanced stateful information for the flows is needed.

Figs. 5 and 6 show the measured accuracy of the heaviest flow selection by the described simple method. These graphs show the relations between the value of threshold $k$ to the portion of heavy marked flows (first graph) and packets covered by them (second graph). By a combination of values from both graphs we can see that with the rising decision threshold the portion of flows marked heavy dramatically decreases, but the percentage of covered packets decreases rather slowly. For example, decision threshold $k = 20$ leads to only 5 percent of heavy marked flows while covering around 85 percent of all packets on average. Exceptions are DNS and to some extent also HTTPS and SMTP protocols, where the percentage of covered packets decreases quickly.

Fig. 7 shows a different view on the simple heavy flow prediction method effectiveness. It shows the average number of packets covered by one heavy marked flow for different values of the decision threshold $k$. Values shown in the graph rise with the decision threshold to a considerably higher number than the average sizes of the flows from Table 1. For example the average size of flow with more than $k = 20$ packets is over 500 packets, while Table 1 reports overall average of 32.0 packets per flow. This clearly proves that even our simple heavy flow prediction method effectively predicts the heaviest flows. Certainly there are many more advanced methods of heavy flow prediction, but these are out of scope of this paper.

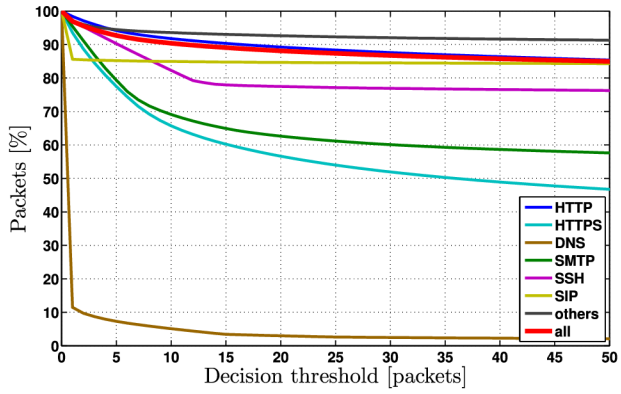


Fig. 6. Heavy flow detection using the simple method—portions of captured packets.

## 3 System Design

Many 10 Gbps flow measurement systems have adopted a common scheme. A hardware network interface card performs packet capture, sometimes enhanced by packet distribution among several CPU cores. The captured traffic is then sent over the host bus to the memory, where packets are processed by the software applications running at the CPU cores [6]. This model cannot be applied to 100 Gbps networks due to major performance bottlenecks. The main bottleneck lies in limited computational power of CPU which is insufficient for advanced monitoring tasks.

We propose a new acceleration model that overcomes the above-mentioned bottlenecks by a well-designed hardware/software system. The main idea is to give the hardware the ability to handle basic traffic processing. Only the control of the HW and advanced processing of a fraction of the traffic are left for the software. Although the preprocessing is done by the firmware in FPGA, it is fully controlled by the software applications. Therefore, the first few packets of each new flow are sent to the software, which selects a type of hardware preprocessing used for the subsequent packets of that flow. Complete software control of the monitoring process is also the reason why we called the proposed model Software Defined Monitoring (SDM).

The types of data preprocessing in the SDM hardware suitable for the area of network monitoring can be divided into three basic groups:

- *Extraction* of the interesting data from packets and sending only those data to the software in a predefined format, which we call a Unified Header (UH). Then only a few bytes for each packet are transferred from hardware to software, thus reducing the PCIe utilization. Also the CPU has lower load, because the packet parsing is done in the hardware.
- *Aggregation* of packets into flow records directly in the hardware, which brings even higher performance savings for the CPU. This aggregation may range

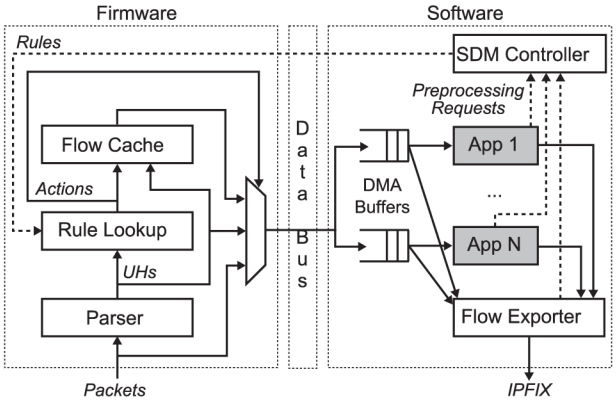


Fig. 8. Conceptual top-level scheme of SDM system.

from basic flow statistics to very specific actions according to the needs of particular applications.

- *Filtration* of unnecessary packets and forwarding only the interesting ones to the software. This can aid advanced monitoring applications, which perform various analyses and detections oriented only to some specific subgroup of network traffic (e.g. DNS threat detector or HTTP header analyzer).

Top-level conceptual scheme of the proposed SDM model is shown in Fig. 8. Forward path is represented by solid arrows and an offload control feedback path by dashed arrows. The system is composed of two main parts (FPGA firmware and software on general CPU) connected together through a data bus. The bus can be PCI Express in case of using commodity PC with a hardware accelerator, or any other interface (e.g. ATCA backplane or internal bus of a single die CPU+FPGA chip).

The processing of all incoming packets starts with parsing a header and extracting packet's metadata (Parser). Extracted metadata is then used to classify the packet based on a software defined set of rules (Rule Lookup). Each rule identifies one concrete flow and specifies the type of packet preprocessing and the target software channel for packets of that flow. Packets can be processed in a firmware flow cache (i.e., aggregated to the selected type of flow record), dropped, trimmed or sent to the software unchanged or in the form of a Unified Header. Flow records residing in the firmware flow cache are periodically exported to the software. The periodic checking is not shown in Fig. 8 for clarity. The data from the firmware is sent over the bus to the software using multiple independent channels. Data for each channel is stored in a software buffer in the form of whole packets, Unified Headers or flow records.

This data is processed by the set of user specific software applications such as the flow exporter [1] which analyzes the received data and exports the flow records to a collector. User applications read the data from the selected channels. They also specify which types of traffic they want to inspect and which flows can be preprocessed in hardware. Definitions of uninteresting traffic from all applications are passed to a software SDM controller daemon. The SDM controller aggregates the definitions (requests) into rules and configures the firmware preprocessing in order to achieve the maximal possible reduction of traffic while preserving the required level of information so that not a single piece of application interesting information is lost. This mechanism realizes the feedback control loop, which is the important concept in our work.

Network traffic preprocessing in the firmware is entirely controlled from the software and the core of the controlling software consists of the monitoring applications (App 1..N). Each monitoring application has the form of a software plugin. The main input to the plugin is the data path carrying the packets, extracted UHs or aggregated flow records. The plugin output is whichever data that the plugin has parsed/detected/measured. This output data is added to the exported IPFIX flow record, so it is *enriched* by the information from the plugin. Each monitoring application also has the interface to the SDM controller.

### 3.1 SDM Controller

SDM controller accepts the preprocessing requests from multiple applications and aggregates them into rules for the firmware. It also manages timed expiration of application requests and periodical export of aggregated flow records from hardware. The aggregation of preprocessing rules is based on different degrees of data reduction. Ordered from the lowest degree of data reduction the preprocessing types are: none (whole packets), trim (shortened packets), partial (UH), complete (flow record) and elimination (packet drops). Therefore, aggregation of rules in the SDM controller is done simply by the selection of the lowest preprocessing degree (highest data preservation) for particular flows which satisfy the information level requirements of all monitoring applications. In order to maintain a proper functionality of the SDM firmware, the controller must carry out the following operations:

- Management of rules activated in the firmware (rule add/delete/update) based on the application demands.
- Decision about offloading particular flows based on the estimated flow size and the free space in the firmware flow cache.
- Cyclic export of active flow records computed in the firmware flow cache.
- Allocation of records in the firmware flow cache.

In the previous section we have presented the method of heavy flow estimation based on the simple packet count threshold. In the design for practical implementation, we further extend this idea by using *adaptive threshold* that automatically reacts to the changing characteristics of network traffic in time. The adaptation is based on current load of the firmware flow cache, which has a limited size. For the best offload ratio, it is advantageous to keep the flow cache nearly full at all times. That way, there is still some space left for the new flows, while the amount of offloaded traffic is maximized. Therefore, SDM controller periodically checks the flow cache state, decreases the heavy flow decision threshold when the flow cache utilization drops below a specified point, and increases the threshold when the flow cache utilization rises.

### 3.2 SDM Firmware

Top level implementation scheme of the SDM accelerator firmware for FPGA is shown in Fig. 9. The main firmware

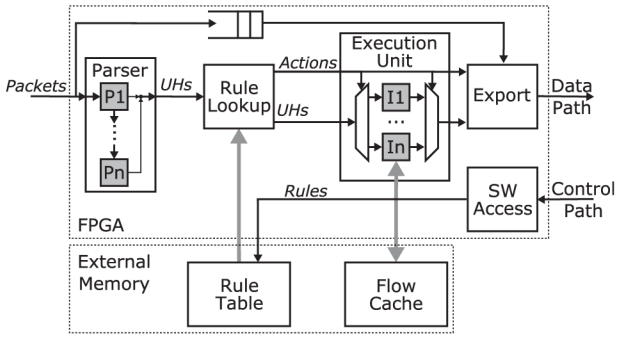


Fig. 9. Detailed firmware scheme.

functionality is realized by a processing pipeline that processes incoming network traffic and creates an outgoing data flow for the software. Packets do not flow directly through the processing pipeline, but are rather stored in a parallel FIFO buffer. The processing pipeline uses only meta-information (UH) extracted from packet headers by Parser. Whole software control of the processing pipeline is realized through SW Access module which conveys the preprocessing rules to be used in Flow Search unit.

The SDM firmware is realized by five main modules:

*Parser* extracts interesting information from headers of packets, especially fields that clearly identify network flows. To identify the flows, we use the five-tuple: IP addresses, TCP/UDP ports and protocol. Furthermore, our implementation is modular and enables easy extensions of default packet parsing process by additional application-specific parser modules (P1..Pn). This way, the information extracted from each packet can be enhanced when required. Further information about this parser can be found in [9].

*Rule Lookup* assigns an action (processing instruction) to every packet based on its flow identifier and a set of software defined rules. Management of the rule set is done through a control interface capable of atomic on the fly add, remove and update of the rules.

*Execution Unit* manages stateful flow records in Flow Cache. It mainly actualizes their values by execution of instructions from flow associated actions. Every action specifies an instruction that should be executed and the address of the flow record to work with. Furthermore, the instruction has access to data extracted from packet (UH). Special type of instruction is an export of the record values, possibly followed by a reset of the record. Records can be exported not only at the flow end but also in a periodical manner, so that the software applications can have actual information about flows in the firmware. Control of memory allocation for records and their periodical export is left to the SDM controller. The Execution Unit supports multiple user-defined instruction sub-modules (I1..In). More details about the execution and implementation of instructions are in Section 3.3.

*Export* pairs together corresponding UH transaction with frame data from FIFO buffer. Then it chooses the required channel and format for the data based on action assigned by Rule Lookup module.

*SW Access* is the main configuration access point into the SDM firmware from the software side. Its primary function is to manage the rules and to initiate export of the flow records based on controller commands. Besides, it contains all configuration and control registers.

### 3.3 Execution Unit Functionality

As already mentioned, Execution Unit realizes the main stateful behavior of the hardware by execution of flow record updating instructions. To improve the overall flexibility of the system, we use modular architecture within the Execution Unit that allows us to implement custom read-modify-write aggregation operations (instructions). Thanks to these custom instructions, the nature of the flow records maintained by the hardware in Flow Cache can be customized according to a target application. Furthermore, we use high-level synthesis (HLS) tools to generate custom hardware modules from an instruction description in C or C++. Thanks to that, SDM hardware functionality can be customized faster and even without the knowledge of HDL programming (e.g. by network security experts). Also an incremental, performance driven design of new hardware accelerated applications is much easier. The process starts with a software implementation of the application, accelerated only by the default SDM instructions. Then the performance bottleneck is identified and the critical piece of code is moved into the FPGA as a new instruction with minimal extra effort.

We have already implemented and evaluated five different Execution Unit instructions to test the feasibility of the described concept with HLS usage:

- *NetFlow* instruction is used for standard NetFlow aggregation. Its execution increases flow packet and byte counters, updates flow end timestamp and computes logical OR of the observed TCP flags.
- *NetFlow Extended* instruction has the same basic functionality as NetFlow. In addition, it stores TCP flags of the first five packets. This additional information may become very useful for analysis of TCP handshake or for detection of network attacks like DoS (Denial of Service).
- *TCP Flag Counters* instruction performs increment of counters of individual 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 [10].
- *Timestamp Diff* instruction maintains records of inter-arrival times of the first 11 packets of the flow. These times have nanosecond precision and can be used as network discriminators for flow-based classification [10] or for identification of application protocol [11].
- *CPD* instruction (Change-Point Detection) shows implementation of more complex operation. CPD is an algorithm designed to detect an anomaly in the processed network flow. Description of this method is out of scope of this paper, more details can be found in [12], [13].

## 4 Results

We have implemented the whole SDM prototype in order to verify the proposed concept. The hardware part of the prototype is realized on an accelerator board with a powerful Virtex-7 H580T FPGA (Fig. 10). The FPGA firmware realizes

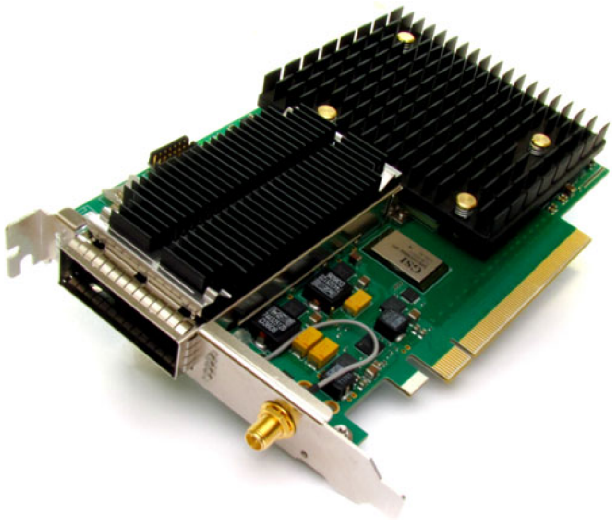

Fig. 10. COMBO-100 G accelerator used for our prototype implementation.

the SDM functionality, such as packet header parsing and NetFlow statistics updating, but also 100 Gbps Ethernet, PCI-Express and QDR external memory interface controllers. The software is realized as a set of plugins for the Invea-Tech's Flowmon exporter software [14]. This exporter allows us to modify its functionality to the extent required by the SDM concept.

We follow by measurement of the real effectivity of the heavy flow detection. Control of the hardware preprocessing is mainly realized by the monitoring applications through on the fly defined dynamic rules for particular flows. These rules are generated as a reaction to the first few packets of the flow. Therefore, there is some delay between the flow start and offload rule application. The duration of this delay influences a portion of packets affected by the rules. The basic view of achievable SDM effectiveness can be gained from an examination of an achievable portion of packets whose preprocessing was influenced by the dynamic flow rules.

We have created a simple use case in order to test the described ability of the SDM concept. In this use case, only a specified number of the first packets from each flow are interesting to the software. All packets from unknown (new) flows are, therefore, forwarded into the software application by default. SDM controller software counts the number of packets in each flow. Right after the reception of the specified number of packets for a flow, the application creates a rule for the firmware to drop all the following packets from this flow. This decision method is absolutely the same as the simple heavy flow detection method defined in the previous section, but the adaptive threshold is not employed in this use case.

We have measured the portion of packets dropped by the SDM firmware in the described test case. The results are projected into the graph in Fig. 11. The graph shows the percentage of dropped (influenced) packets (solid lines) and the percentage of flows for which the rule was created (dashed lines). For comparison, analytical results from graphs 5 and 6 in the previous section are also shown (red). The result is that the SDM can influence preprocessing of

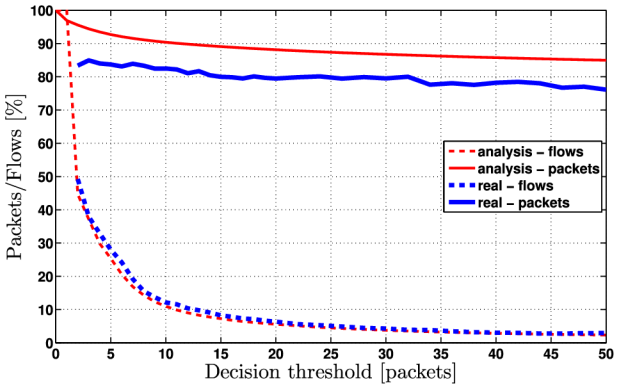


Fig. 11. Portions of offloadable packets and flows using the simple heavy flow detection method.

up to 85 percent of all packets from real network traffic by dynamic flow rules. A visible difference of about 10 percent of influenced packets between analytical and real results is caused by neglecting the duration of rule creation and activation process in the analytical result.

The portions of offloaded packets and flows are similar to the analysis in Section 2—there is a considerably faster decline in the percentage of flows than in the percentage of packets. A different view is provided in Fig. 12. There, a relation of the mean number of packets influenced by one created rule over the decision threshold value is shown (blue). The red line is analytical result of simple heavy flow detection method effectiveness taken from Fig. 7. The graph shows that real measured effectiveness of this method is slightly worse than the analysis suggests, but still suitable for real usage.

We also provide a test of SDM acceleration abilities in more realistic use cases. We test the performance of the concept prototype in the following four cases:

- *Standard NetFlow measurement.* In this use case, all packets from a network line are taken into account. Since NetFlow measurement is based on counting statistics of packet headers only, the packets are sent to the software in the form of UH by default. The software then adds dynamic rules to offload the NetFlow measurement of heavy flows (predicted by our simple method) into the hardware accelerator.

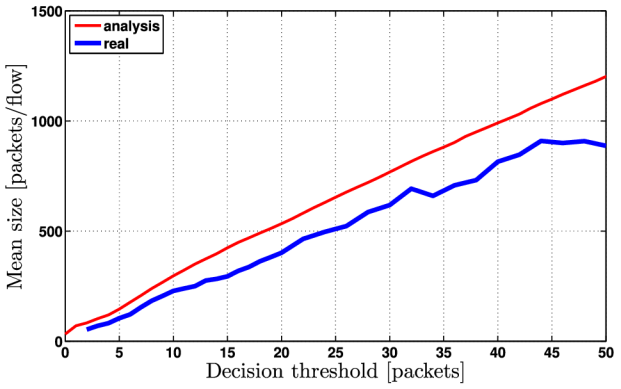


Fig. 12. Mean number of offloadable packets per flow in flows selected using the simple heavy flow detection method.

TABLE 2
Usage of Hardware Preprocessing

| Use case | Preprocessing method [% of packets] | | | |
|---|---|---|---|---|
| | Packet | Header | NetFlow | Drop |
| NetFlow | – | 20.55 | 79.45 | – |
| Port scan | – | 17.54 | – | 82.46 |
| Heartbleed | 4.91 | – | – | 95.09 |
| HTTP | 22.82 | – | – | 77.18 |
| HTTP+NetFlow | 23.34 | 10.56 | 66.10 | – |

- *Port scan detection.* This use case demonstrates a measurement that is flow-based, yet not directly NetFlow. The software plugin observes UHs of first several packets of each flow and installs drop rules for the subsequent packets of heavy flows. This information is typically enough to detect port scan attacks through various methods.
- *Heartbleed detection*. clearly demonstrates the need for application-layer processing in the network security monitoring. The software application first instructs the accelerator to drop all non-SSL packets (i.e., other than TCP port 443). Then further rules to drop packets of heavy SSL flows are installed in the runtime, because the Heartbleed attack can be detected by observing first few packets of each flow.
- *HTTP header analysis.* From application layer protocols we choose HTTP because our network analysis in Section 2 shows that HTTP traffic is dominant in current networks. Therefore, acceleration of its analysis is of high importance. In this use case we test an application that parses HTTP headers and extracts some interesting information (e.g. URL, host, user-agent) from them. Because the application works with the data of HTTP packets, only the packets with a source or destination port 80 are sent into the software by default. Others are dropped in the hardware. Furthermore, the application adds dynamic rules to drop the packets of HTTP flows in which it already detected and parsed the HTTP header.
- *Standard NetFlow enriched by HTTP analysis.* This case combines two of the previous use cases. Both NetFlow exporter and HTTP parser are active at the same time without the need of any changes in them. Their traffic preprocessing requirements are automatically combined by the SDM controller.

Tables 2 and 3 show the results of the SDM system testing in the described use cases. The tables show portions of all incoming packets and bytes preprocessed in the hardware by each preprocessing method. These hardware preprocessing utilizations lead to a reduction of software application load displayed in Table 4. The table shows portions of incoming packets and bytes that are processed by software applications in each use case relative to the state without the SDM accelerator. It also shows a percentage of flows for which a rule was created in the hardware.

TABLE 3
Usage of Hardware Preprocessing

| Use case | Preprocessing method [% of bytes] | | | |
|---|---|---|---|---|
| | Packet | Header | NetFlow | Drop |
| NetFlow | – | 12.03 | 87.97 | – |
| Port scan | – | 10.35 | – | 89.65 |
| Heartbleed | 3.77 | – | – | 96.23 |
| HTTP | 27.82 | – | – | 72.18 |
| HTTP+NetFlow | 28.50 | 3.63 | 68.87 | – |

TABLE 4
Software Applications Load Using SDM in Tested Use Cases, Relative to the State without the SDM Accelerator

| Use case | SW load [%] | | Flows covered by rules [%] |
|---|---|---|---|
| | Packets | Bytes | |
| NetFlow | 20.66 | 0.98 | 6.37 |
| Port scan | 17.54 | 0.86 | 6.53 |
| Heartbleed | 4.91 | 3.77 | 0.95 |
| HTTP | 22.82 | 27.82 | 1.98 |
| HTTP+NetFlow | 34.02 | 29.00 | 6.04 |

Standard NetFlow measurement is significantly accelerated by the hardware flow cache. In this way, the software application load is reduced to one fifth of all packets (in the form of UH or flow record). Further acceleration rises from the fact that only UHs and flow records are sent to the software, instead of complete packets. The software, therefore, does not parse packets anymore and the PCI Express bus load is reduced to less than one percent.

The unnecessary packets are dynamically dropped in the Port scan scenario. Furthermore, the detector do not require whole packets—UHs are sufficient. This constellation leads to considerable savings of both bus bandwidth and CPU load.

Dropping the packets based on static and dynamic rules is also the preferred method of acceleration in both Heartbleed detection and HTTP protocol analysis scenarios. This leads to the HTTP parser load being reduced to only about a quarter of all packets and bytes and even more significant reduction in the Heartbleed detection. Due to the fact that both static and dynamic rules are used, the percentage of dropped packets is split in two parts. In the HTTP use case 51.84 percent of all packets were dropped by a static TCP port 80 check, and 21.34 percent of packets belonged to heavy TCP port 80 flows for which the dynamic rule has been installed by the SDM controller.

In the standard NetFlow measurement together with the application protocol parsing scenario, the load of the application protocol parser is the same as when used alone thanks to the DMA channel traffic splitting supported by SDM. The HTTP parser software still receives only the packets on the TCP port 80. The load of the software NetFlow measurement slightly rises compared to the NetFlow only measurement, because of the packets that are sent to the software for the HTTP analysis (NetFlow measurement sees also the HTTP packets).

Graphs in Figs. 13, 14, and 15 show results of SDM prototype testing in the NetFlow use case in more details. In the graphs we can see courses of various parameters of SDM system during whole day of NetFlow measurement. Packets preprocessing ability of the accelerator is presented in the first graph. During the whole day, the majority of all

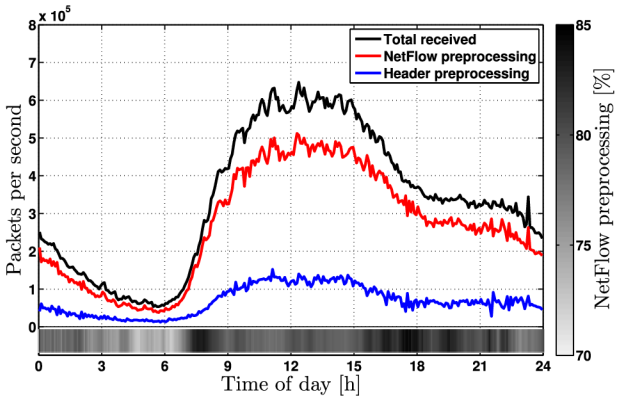


Fig. 13. 24 hours NetFlow measurement with SDM—processed packets.

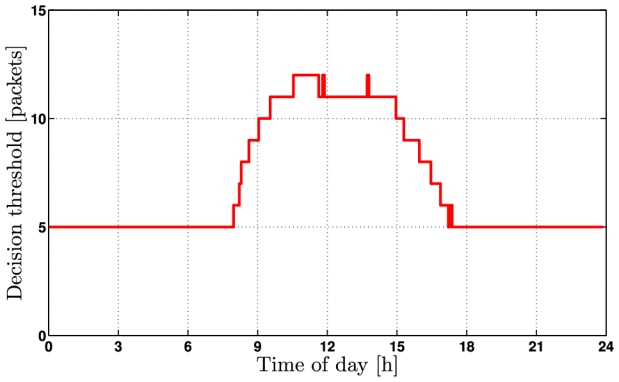


Fig. 15. 24 hours NetFlow measurement with SDM—decision threshold.

received packets (black line) are processed in the firmware flow cache (red line), leaving only a small portion for software processing (blue line). Offloaded percentage of packets is always in the range from 70 to 85 percent of total traffic and is shown in gray shade bar at the bottom of the grid. The second graph shows the number of active rules maintained in the SDM firmware compared to the number of active flows in the network. Black dashed lines demarcate a desired flow cache load maintained by the adaptation of heavy flow decision threshold. There is a significant spike in the total number of flows in the network at around 10:55 pm. After further analysis, we have found that the spike was caused by a mid-sized DoS attack with randomly generated port numbers. Each attacking packet represented a separate flow and was therefore not offloaded to the accelerator. That is a desired behavior, since we want to retain as much information about the attack as possible.

The adaptation of the threshold value during the measurement is illustrated in the third graph. During heavy network load, the threshold value raises to keep the number of offloaded flows within the given range. When the load starts to decline at around 3 pm, the threshold value follows until it reaches a chosen reasonable minimal value (five packets).

For the NetFlow use case, we have also measured a SDM performance curve after system startup in heavy network traffic. The results are depicted in Fig. 16. At the start of the test, the SDM functionality is disabled and all packets are sent for processing into CPU (0 % offload). When SDM is enabled (time 0), we immediately see quick increase in the percentage of offloaded packets as the accelerator is swiftly learning the active heavy flows from the software controller. Around one minute mark, the rise starts to slow down, but still steadily continues for 4 more minutes. After that, the SDM performance is stabilized. Described trend of SDM startup performance curve is very similar also in other tested use cases.

Finally, in Fig. 17 we examine the trade-off in CPU load, since the management of rules in SDM controller represents an additional load to the CPU. We show the effect of SDM acceleration on CPU utilization savings. For this purpose we use the most difficult of our use cases—Netflow measurement together with HTTP analysis. Left half of the graph in Fig. 17 shows measured CPU load with enabled SDM in a stabilized state, right half shows CPU load after SDM was disabled (all processing starts to be done on CPU). According to Table 4, the software load in HTTP +NetFlow use case is up to one third of received packets and bytes when using SDM. This perfectly corresponds to the observed increase in CPU load for packet processing (red line) from 20 to 60 percent after SDM disabling. However, when SDM functionality is enabled, the SDM controller brings some additional overhead (blue line) for configuring the accelerator and aggregating the applications requests. In the end then, total CPU load is two-times lower when using SDM in HTTP+NetFlow use case (black line). This graph also suggests that SDM is best suited for highly

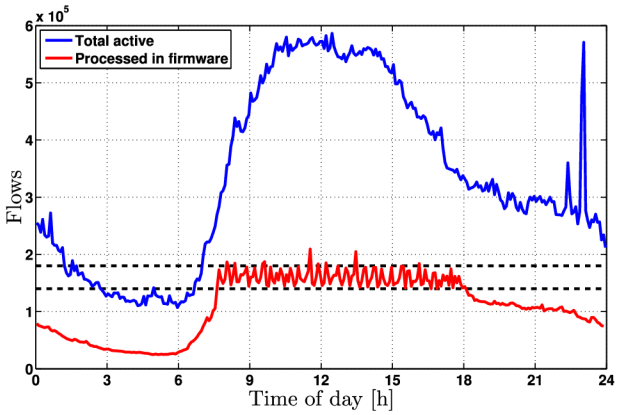


Fig. 14. 24 hours NetFlow measurement with SDM—active rules.

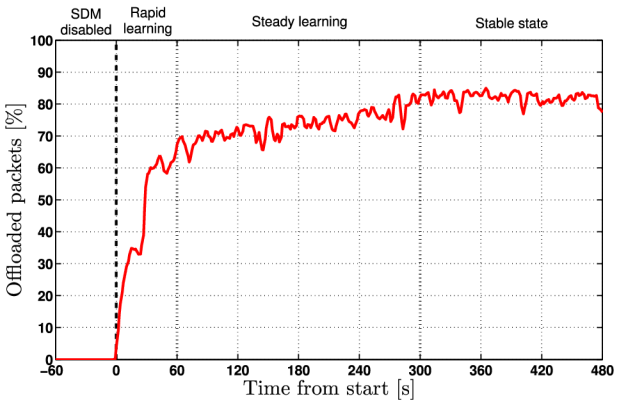


Fig. 16. SDM performance after heavy duty startup.

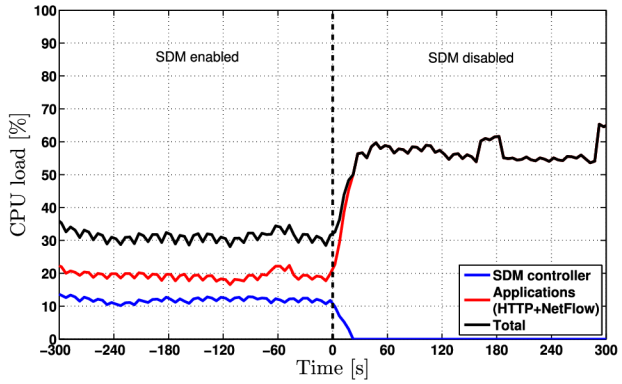


Fig. 17. CPU load in HTTP+NetFlow use case with and without SDM support.

advanced software tasks which consume significant CPU resources. Due to the fact that SDM controller CPU load (blue line) is independent on the application, its share in the total CPU load decreases with the complexity of the application (red line).

### 4.1 FPGA Implementation Results

Our high-speed SDM FPGA firmware runs at 200 MHz and occupies less than half of the available FPGA (Virtex-7 H580T) resources. Closer look at the FPGA resources of the firmware is shown in Table 5. Using the same SDM core with a data width of 512 bits and throughput of 100 Gbps, we have created three different FPGA architectures for boards with three different arrangements of Ethernet ports: one 100 GbE port, two 40 GbE ports and eight 10 GbE ports.

In addition to the high-performance 100 Gbps solution, we also provide an analysis of the SDM core with narrower data width. These solutions can be used in applications with lower throughput requirements, e.g. in embedded 1 or 10 Gbps probes. Note that the results for data widths other than 512 bits were obtained by simple downscaling of the SDM core. Further optimizations are certainly possible to achieve significantly lower FPGA resource utilization for lower throughputs.

Table 6 shows the resource utilization of the individual instruction sub-modules for the Execution Unit. It can be seen that the additional instruction sub-modules are relatively small, compared to the whole firmware, and therefore adding new instruction should not involve any major refinements of the FPGA firmware. Furthermore, a comparison between high-level synthesis and handmade implementation can be seen from the first two rows of the table. Handmade implementation occupies less than a half of LUTs and a bit less registers compared to HLS result. On the other hand, the creation of C implementation of the instruction and its subsequent automatic synthesis to HDL is much faster and simpler than HDL implementation.

TABLE 5
Resources of the SDM Firmware

| Firmware/Module | | Regs | LUTs | Throughput |
|---|---|---|---|---|
| Complete SDM | | 197,758 | 249,214 | 1 × 100 Gbps |
| | | 134,172 | 178,984 | 2 × 40 Gbps |
| | | 184,084 | 222,745 | 8 × 10 Gbps |
| SDM core | 512 b | 30,497 | 51,333 | 100 Gbps |
| | 256 b | 25,866 | 42,793 | 50 Gbps |
| | 128 b | 23,534 | 39,006 | 25 Gbps |
| | 64 b | 22,384 | 37,233 | 12.5 Gbps |
| | 32 b | 21,908 | 36,803 | 6.25 Gbps |
| Virtex-7 H580T FPGA | | 725,600 | 362,800 | |

TABLE 6
Resources of the Instruction Blocks

| Instruction | Regs | LUTs |
|---|---|---|
| NetFlow (handmade VHDL) | 1,754 | 325 |
| NetFlow | 1,846 | 824 |
| NetFlow Extended | 2,070 | 1,113 |
| TCP Flag Counters | 0 | 1,046 |
| Timestamp Diff | 5,199 | 2,556 |
| Change-Point Detection | 5,296 | 3,919 |

## 5 Related Work

We discuss several approaches that may to some extent resemble the SDM concept. However, we show that our work has significant differences to those works.

Snort [15] is an open source software network intrusion prevention and detection system. It relies heavily on regular expression matching, while our work does not enforce nor assume any particular type of software processing. While many papers dealing with hardware acceleration of Snort have been published, they typically restrict their focus to regular expression matching only. We argue that network security monitoring is much more complex task than that and the limitation to RE matching makes those systems unfeasible for practical use. L7-filter [16] is a Linux packet classifier software aiming at protocol identification. It resembles Snort as it also relies on regular expressions.

A good example of a complex software library for application layer traffic processing (showing that RE matching is not sufficient) is nDPI [17]. While this open source library is probably too complex to be hardware accelerated, we envision that similar software can be used as a basis of a SDM plugin.

The OpenSketch architecture [18] defines a configurable pipeline of hashing, classification and counting stages. These stages can be configured to perform the computation of various statistics. OpenSketch is tailored to compute *sketches*—probabilistic structures allowing to measure and detect various aspects of the network communication with a defined error rate. It is not intended for complete NetFlow-like monitoring, nor for exact, error-free measurements. Also, OpenSketch does not allow for application level protocol parsing.

FlowContext system [19] provides a flexible way to implement stateful network traffic processing in an FPGA. NetFlow monitoring is among the examples of its use. However, it does not provide tight control feedback loop to a software application, and therefore cannot be effectively used for problems exceeding the capabilities of a single FPGA.

There have been efforts to implement NetFlow traffic monitoring in FPGAs, most recently even as an open source project [20] for the NetFPGA platform. Our work is however more flexible by allowing application protocol processing in

the software and further acceleration through extensions of the Execution Unit.

The Shunt system [21] is a hardware accelerator with support to divert a suspicious/interesting traffic to a software for further analysis. To this end it resembles our work, however, Shunt accelerates only packet forwarding and does not include any possibilities to offload/accelerate the flow measurement tasks. Our work is also more complete by defining the software architecture with the plugin support.

Xilinx has recently announced SDNet [22] environment for software defined, hardware accelerated networking. The system uses high level language(s) to describe a network application, which is then compiled to a form of hardware accelerator for a Xilinx FPGA. From the limited information available at the time of writing, we envision that SDNet could be used to improve SDM by custom application parsers or instruction modules.

The proposed arrangement of SDM resembles OpenFlow [23]: Packets of an unknown flow are passed from a data path to a control software, which in turn may choose to install processing rules into the data path. Similar to plugins for an OpenFlow controller, SDM is also designed to support various software plugins. In addition to that, newer versions of OpenFlow standard define monitoring primitives for the data path. The main difference with OpenFlow is that, for the sake of performance, our system is not distributed, but our controller is rather very tightly coupled with the hardware accelerator—within the same box, or even at the same chip. That allows implementing applications which would be impractical when built as a distributed system. We also propose user-defined modifications to the data plane through the modular Execution Unit—a concept that is unparalleled in OpenFlow. Our system is an instance of Software Defined Networking in a broader sense, yet it is different from OpenFlow.

## 6 Conclusion

We propose a new concept of application level flow monitoring acceleration called Software Defined Monitoring. The concept is able to support application level monitoring and high-speed flow measurements at speeds over 100 Gbps at the same time. Our system focuses on a high speed and high quality flow based measurement with the support of a hardware accelerator. The accelerator is fully controlled by the software feedback loop and offloads the simple monitoring tasks of bulk, uninteresting traffic. The software, on the other hand, decides about the traffic processing on a per-flow basis and performs the advanced monitoring tasks such as application protocol parsing. The software works with monitoring plugins, therefore, SDM is *by design* ready for extensions by new high-speed monitoring tasks without the need to modify its hardware. Moreover, the FPGA accelerator itself can also be improved to support new types of offload.

Our detailed analysis of the backbone network traffic parameters demonstrates the feasibility of the concept. We have also implemented the whole SDM system using the Virtex-7 FPGA accelerator board, including some extensions to the firmware offload engine. The system is ready to handle 100 Gbps traffic. Using the SDM prototype, we have evaluated several use cases for SDM. It is clear from the obtained results that SDM is able to offload a significant part of network traffic to the hardware accelerator and therefore to support a much higher throughput than a pure software solution. The results show a major speed-up in all test cases.

## Acknowledgments

This research has been partially supported by the "CESNET Large Infrastructure" project no. LM2010005 funded by the Ministry of Education, Youth and Sports of the Czech Republic, the research programme MSM 0021630528, the grant BUT FIT-S-14-2297 and the IT4Innovations Centre of Excellence CZ.1.05/1.1.00/02.0070. V. Puš is the corresponding author.

## References

[1] B. Claise, "Cisco systems netflow services export version 9," RFC 3954, Internet Engineering Task Force, Oct. 2004.
[2] B. Claise, B. Trammell, and P. Aitken, "Specification of the IP flow information export (IPFIX) protocol for the exchange of flow information," RFC 7011, Internet Engineering Task Force, Sep. 2013.
[3] L. Deri, L. Trombacchi, M. Martinelli, and D. Vannozzi, "A distributed dns traffic monitoring system," in *Proc. 8th Int. Wireless Commun. Mobile Comput. Conf.*, 2012, pp. 30–35.
[4] M. Elich, P. Velan, T. Jirsik, and P. Celeda, "An investigation into teredo and 6to4 transition mechanisms: Traffic analysis," in *Proc. 38th Conf. Local Comput. Netw. Workshops*, 2013, pp. 1018–1024.
[5] P. Velan, T. Jirsik, and P.Čeleda, "Design and evaluation of http protocol parsers for ipfix measurement," in *Proc. Adv. Commun. Netw.*, 2013, vol. 8115, pp. 136–147.
[6] F. Fusco and L. Deri, "High speed network traffic analysis with commodity multi-core systems," in *Proc. 10th ACM SIGCOMM Conf. Internet Meas.*, 2010, pp. 218–224.
[7] C. Estan and G. Varghese, "New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice," *ACM Trans. Comput. Syst.*, vol. 21, no. 3, pp. 270–313, Aug. 2003.
[8] K.-C. Lan and J. Heidemann, "A measurement study of correlations of internet flow characteristics," *Comput. Netw.*, vol. 50, no. 1, pp. 46–62, Jan. 2006.
[9] V. Puš, L. Kekely, and J. Kořenek, "Low-latency modular packet header parser for FPGA," in *Proc. 8th ACM/IEEE Symp. Arch. Netw. Commun. Syst.*, 2012, pp. 77–78.
[10] A. W. Moore, D. Zuev, and M. L. Crogan, "Discriminators for use in flow-based classification," Department of Computer Science, Queen Mary University of London, London, U.K., Tech. Rep. RR-05-13, 2005.
[11] P. Piskac and J. Novotny, "Using of time characteristics in data flow for traffic classification," in *Managing Dynamics Netw. Serv.*, 2011, vol. 6734, pp. 173–176.
[12] A. Tartakovsky, A. Polunchenko, and G. Sokolov, "Efficient computer network anomaly detection by changepoint detection methods," *Sel. Topics Signal Process.*, vol. 7, no. 1, pp. 4–11, 2013.
[13] R. B. Blazek, H. Kim, B. Rozovskii, and A. Tartakovsky, "A novel approach to detection of "denial of service" attacks via adaptive sequential and batch-sequential change-point detection methods," in *Proc. 2nd IEEE Workshop Syst., Man, Cybernetics*, 2001, pp. 220–226.
[14] INVEA-TECH a.s.. (2014). FlowMon Exporter-Community Program. [Online]. Available: http://www.invea.cz
[15] Snort. (2014). [Online]. Available: http://www.snort.org
[16] l7-filter. (2014). [Online]. Available: http://l7-filter.clearfoundation.com/
[17] ntop, nDPI. (2014). [Online]. Available: http://www.ntop.org/products/ndpi/
[18] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with opensketch," in *Proc. 10th USENIX Conf. Networked Syst. Des. Implementation*, 2013, pp. 29–42.
[19] M. Košek and J. Kořenek, "Flowcontext: Flexible platform for multigigabit stateful packet processing," in *Proc. Int. Conf. Field Programm. Logic Appl.*, 2007, pp. 804–807.

[20] M. Forconesi, G. Sutter, S. Lopez-Buedo, and J. Aracil, "Accurate and flexible flow-based monitoring for high-speed networks," in *Proc. 23rd Int. Conf. Field Programm. Logic Appl.*, Sep. 2013, pp. 1–4.
[21] N. Weaver, V. Paxson, and J. M. Gonzalez, "The shunt: An FPGA-based accelerator for network intrusion prevention," in *Proc. 15th Int. Symp. Field Programm. Gate Arrays*, 2007, pp. 199–206.
[22] Xilinx Inc., SDNet. (2014). [Online]. Available: http://www.xilinx.com/applications/wired-communications/sdnet.html
[23] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

**Lukáš Kekely** is working towards the PhD degree at the Faculty of Information Technology, Brno University of Technology since 2013 and also a researcher at CESNET since 2011. His research is focused mainly on hardware accelerated solutions for high-speed networks, particularly in the area of monitoring and security. He is an author of several research papers published at renowned international conferences.

**Viktor Puš** received the PhD degree from the Faculty of Information Technology, Brno University of Technology in 2012. He is a researcher with focus on hardware acceleration of timing-critical operations in the network, particularly in the network security monitoring. He is an author of one US patent and many research papers published at renowned international conferences.

**Jan Kořenek** received the PhD degree in 2010 from the Brno University of Technology, Czech Republic. He has substantial experiences in the hardware acceleration of network applications which was obtained by working on a number of European and locally funded projects. He is an author of many papers and novel hardware architectures. In May 2007, he co-founded INVEA-TECH which is a successful spin-off company focused on high speed network monitoring and security applications. In 2009, he formed Accelerated Network Technologies (ANT) research group at Brno University of Technology.

**Jan Kučera** graduated with a bachelor's degree at the Faculty of Information Technology, Brno University of Technology in 2014. He continues his studies in the follow-up master's degree programme. Since 2012, he works as a researcher at CESNET and is interested in FPGA design, especially in hardware acceleration for the purpose of high-speed networks monitoring.

**Athanasios V. Vasilakos** served or is serving as an editor or/and guest editor for many technical journals,such as the *IEEE Transactions on Network and Service Management*; *IEEE Transactions on Cloud Computing*, *IEEE Transactions on Information Forensics and Security*, *IEEE Transactions on Cybernetics*; *IEEE Transactions on Nanobioscience*; *IEEE Transactions on Information Technology in Biomedicine*; *ACM Transactions on Autonomous and Adaptive Systems*; the *IEEE Journal on Selected Areas in Communications*. He is also general chair of the European Alliances for Innovation (www.eai.eu ).

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.