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, Božetêchova 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

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

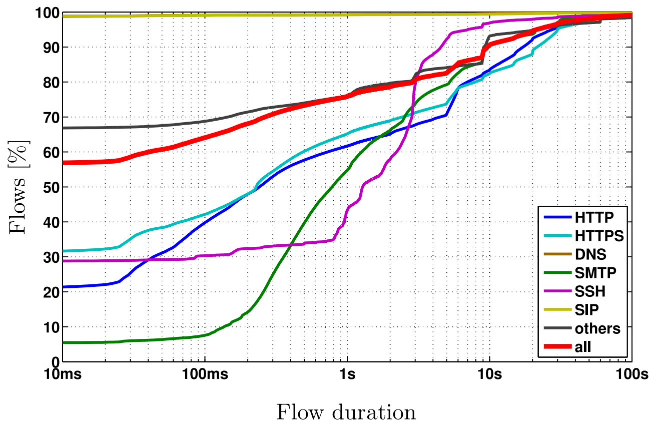


Fig. 1. Cummulative distribution functions of flow durations.

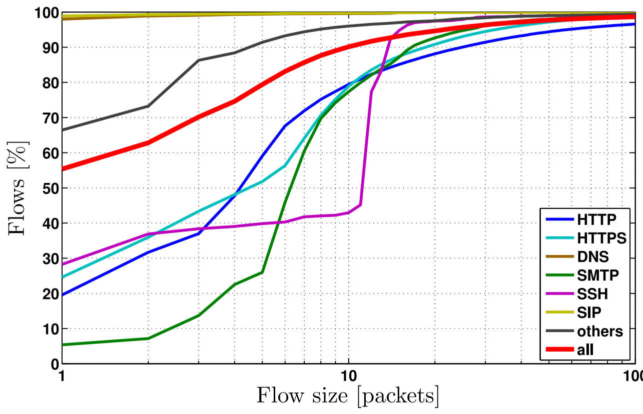


Fig. 3. Cummulative distribution functions of flow sizes.

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

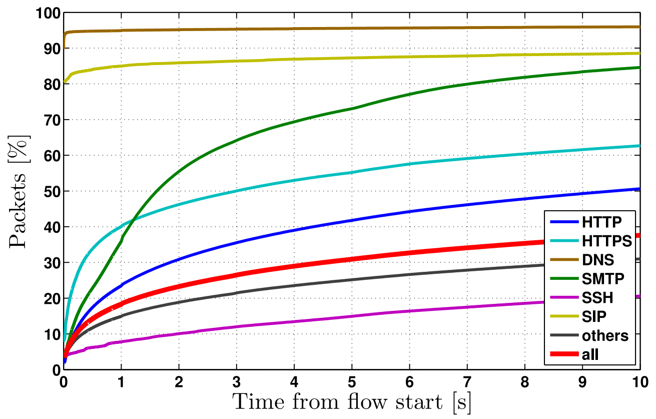


Fig. 2. Cummulative distribution functions of packet arrival times.

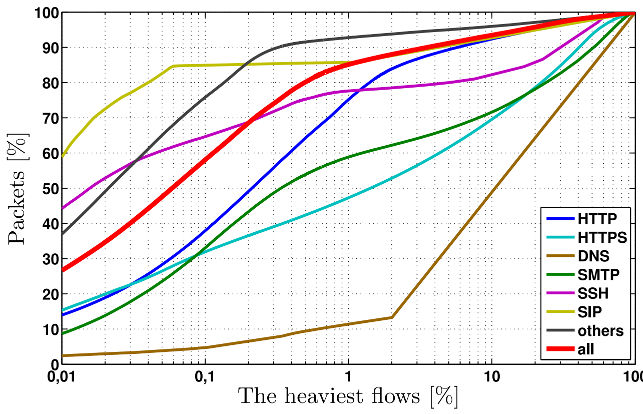


Fig. 4. Portions of packets carried by the percentage of the heaviest flows.

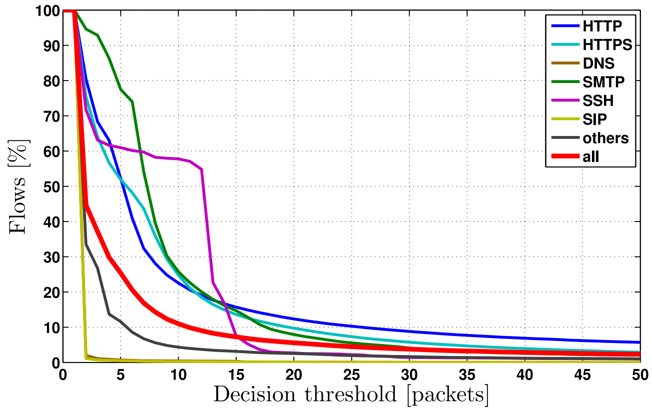


Fig. 5. Heavy flow detection using the simple method—portions of selected flows.

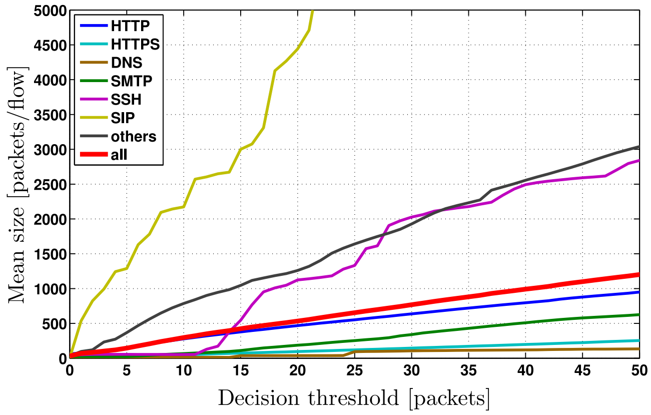


Fig. 7. Mean number of captured packets per flow in flows selected using the simple method.

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

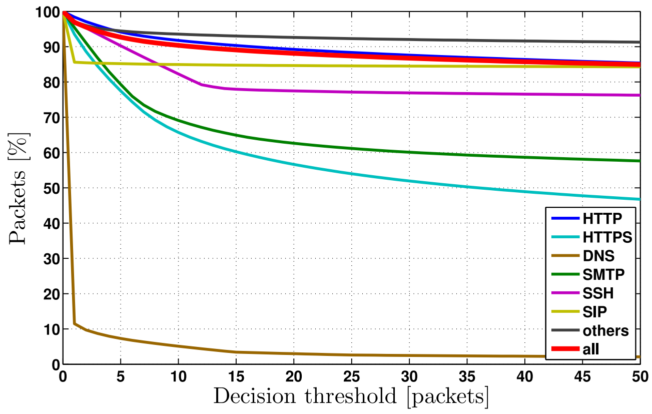


Fig. 6. Heavy flow detection using the simple method—portions of captured packets.

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.

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

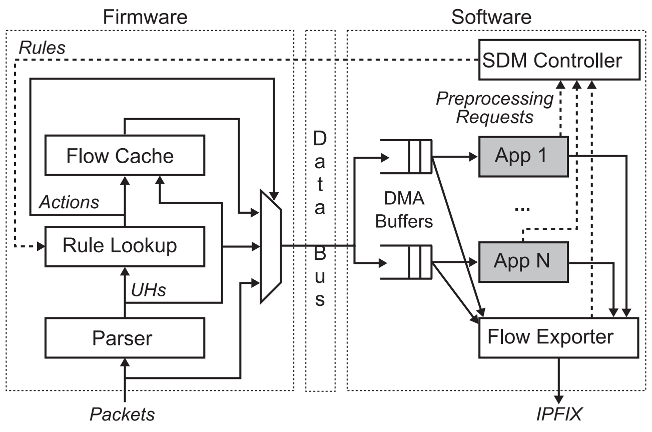


Fig. 8. Conceptual top-level scheme of SDM system.

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

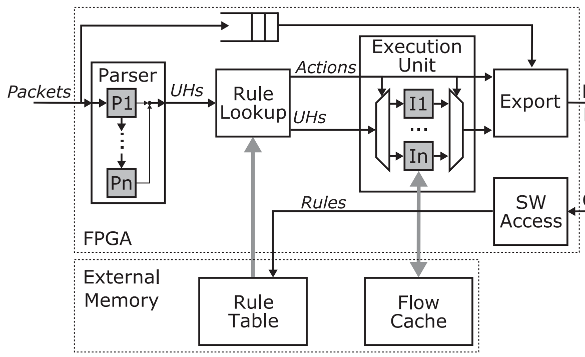


Fig. 9. Detailed firmware scheme.

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 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.

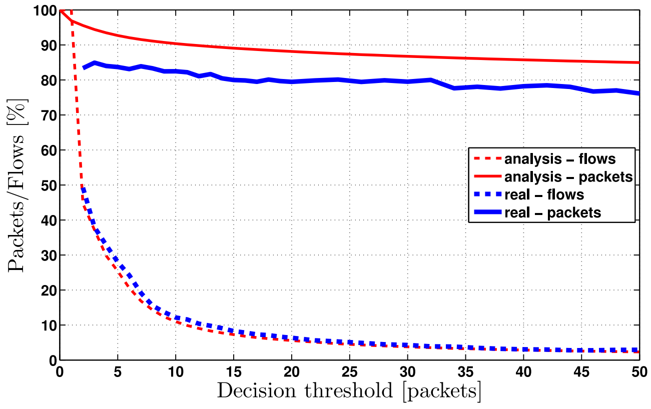


Fig. 11. Portions of offloadable packets and flows using the simple heavy flow detection method.

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.

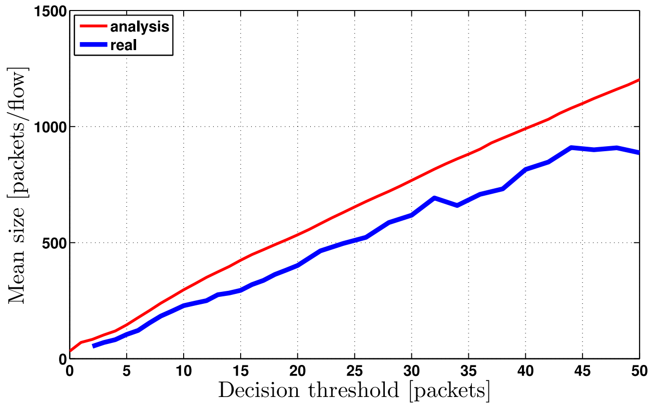


Fig. 12. Mean number of offloadable packets per flow in flows selected using the simple heavy flow detection method.

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

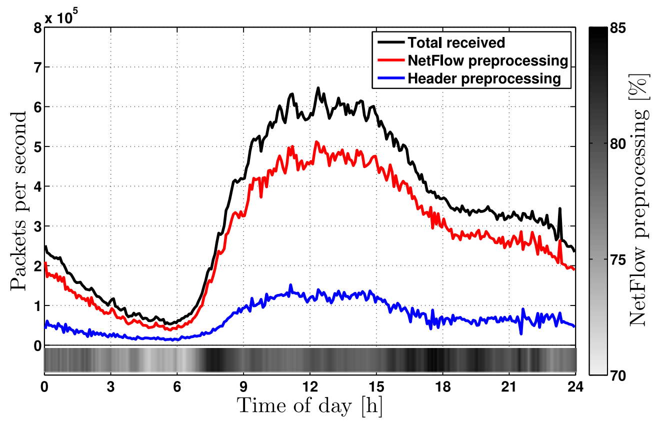


Fig. 13. 24 hours NetFlow measurement with SDM—processed packets.

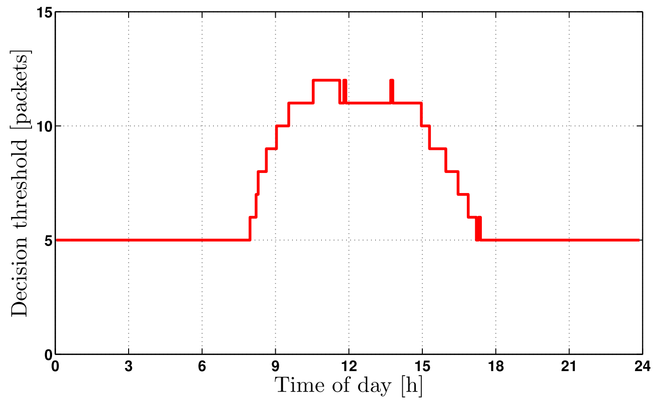


Fig. 15. 24 hours NetFlow measurement with SDM—decision threshold.

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

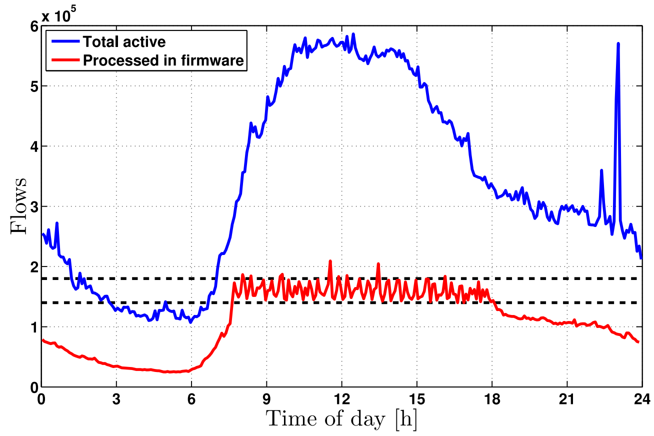


Fig. 14. 24 hours NetFlow measurement with SDM—active rules.

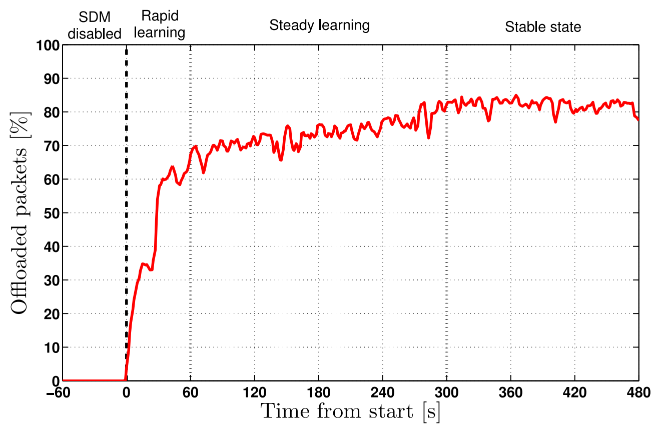


Fig. 16. SDM performance after heavy duty startup.

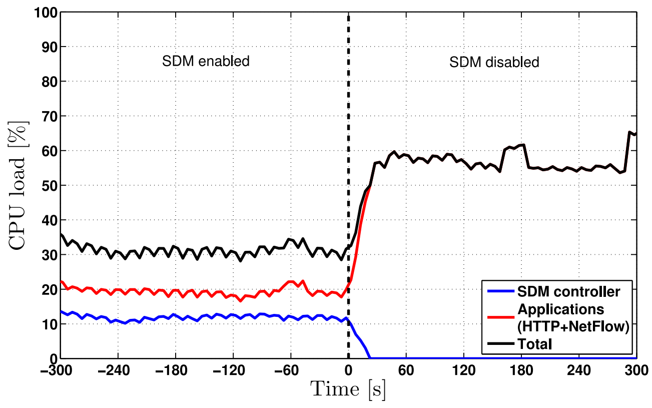


Fig. 17. CPU load in HTTP+NetFlow use case with and without SDM support.

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.

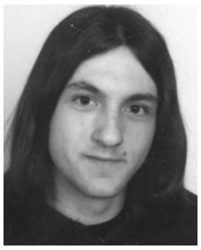
[19] M. Košek and J. Kořenek, "Flowcontext: Flexible platform for multigigabit stateful packet processing," in *Proc. Int. Conf. Field Programm. Logic Appl.*, 2007, pp. 804–807.

[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).