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.

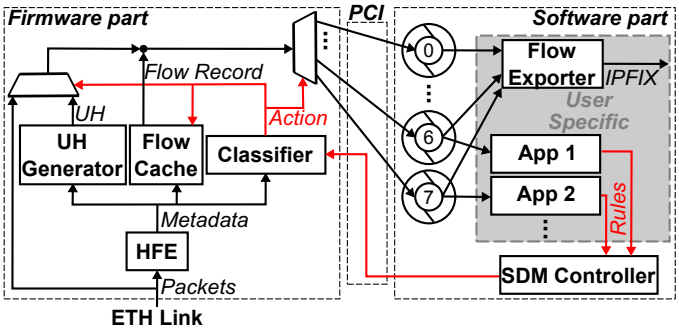

Fig. 1. Conceptual top-level scheme of SDM system

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.

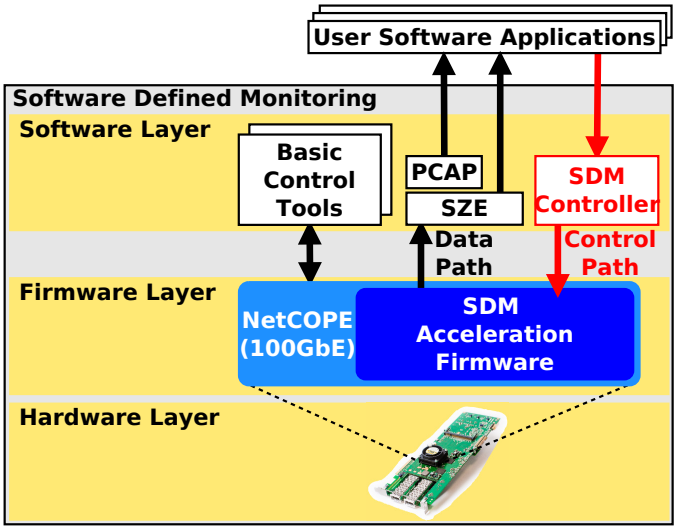

Fig. 2. Layered model of SDM acceleration system

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.

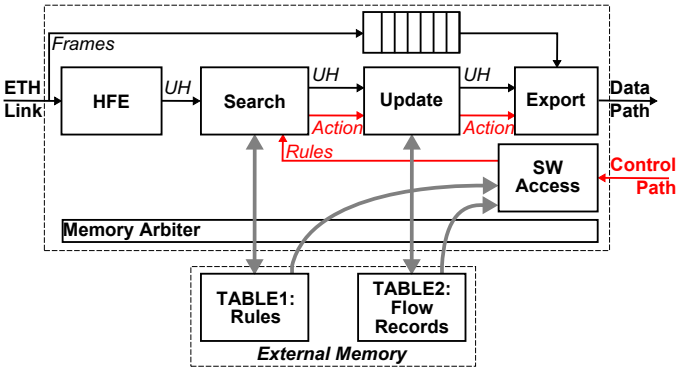

Fig. 3. Detailed firmware scheme

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

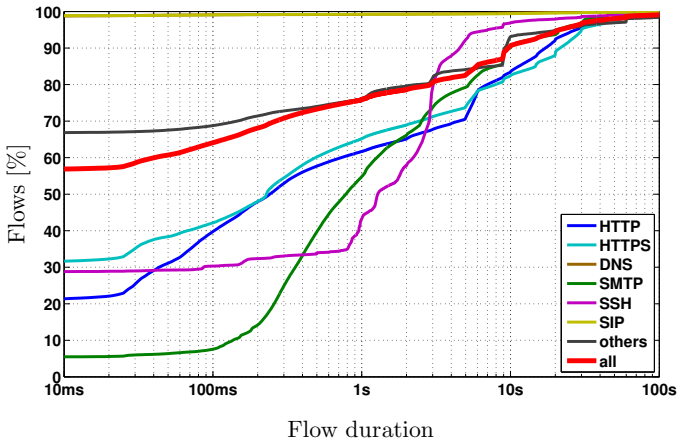

Fig. 4. Cumulative distribution functions of flow durations

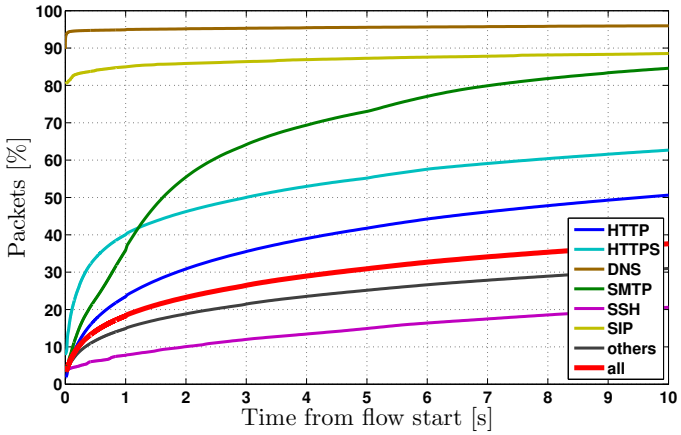

Fig. 5. Cumulative distribution functions of packet arrival times

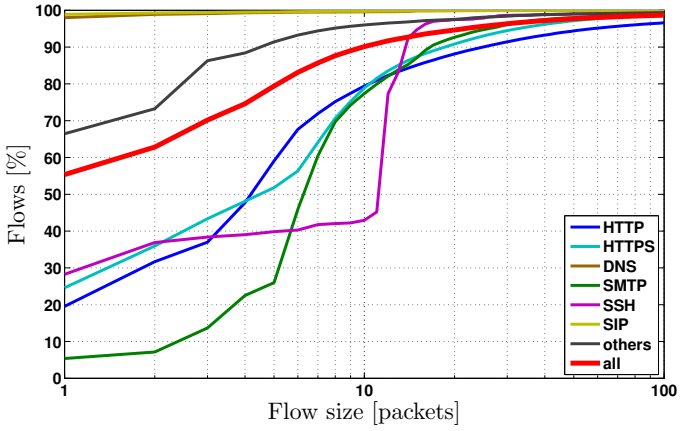

Fig. 6. Cumulative distribution functions of flow sizes

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.

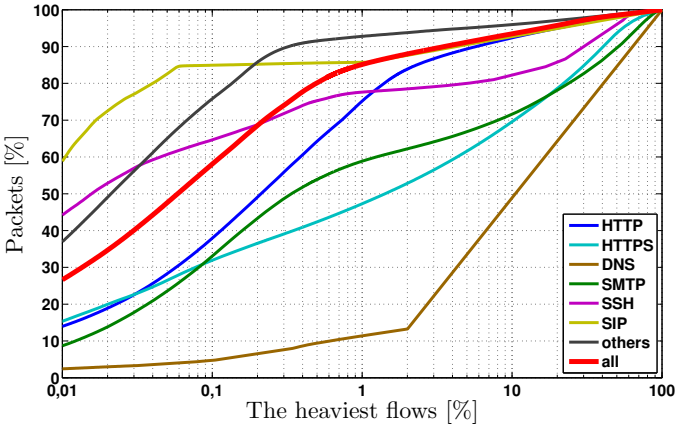

Fig. 7. Portions of packets carried by the percentage of the heaviest flows

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

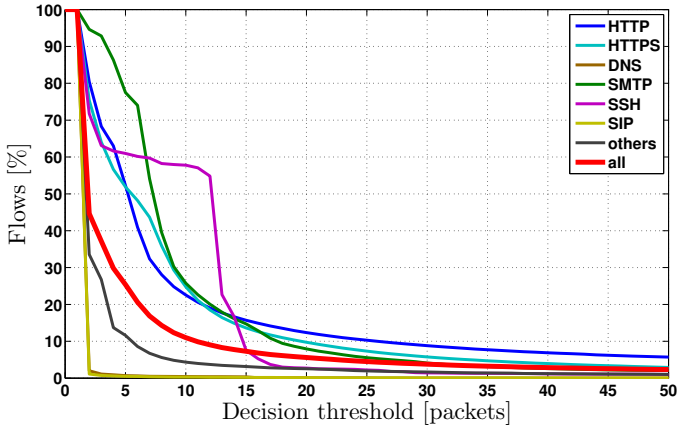

Fig. 8. Heavy flow detection using the simple method – portions of selected flows

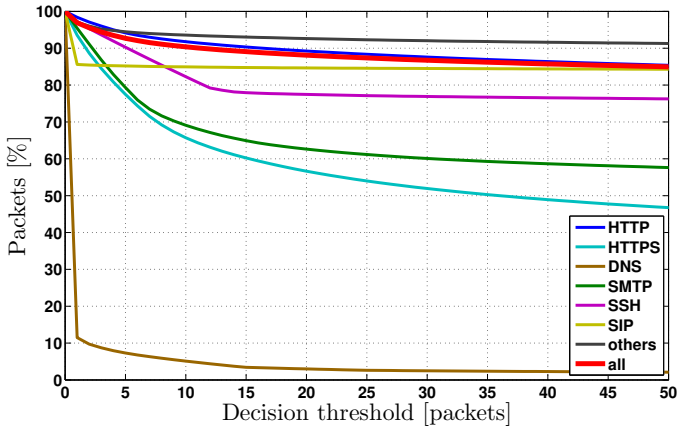

Fig. 9. Heavy flow detection using the simple method – portions of captured packets

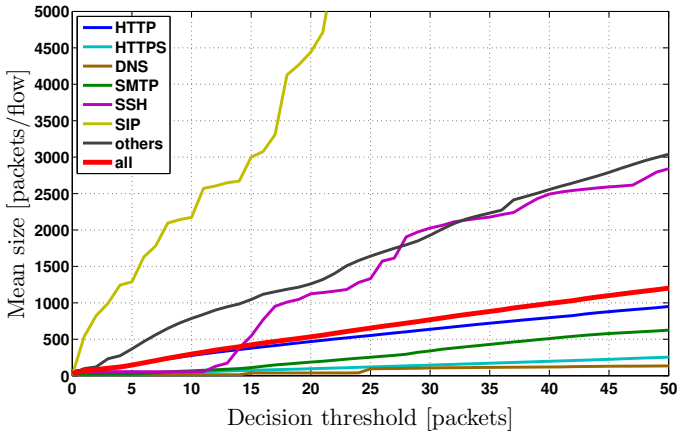

Fig. 10. Mean number of captured packets per flow in flows selected using the simple method

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.

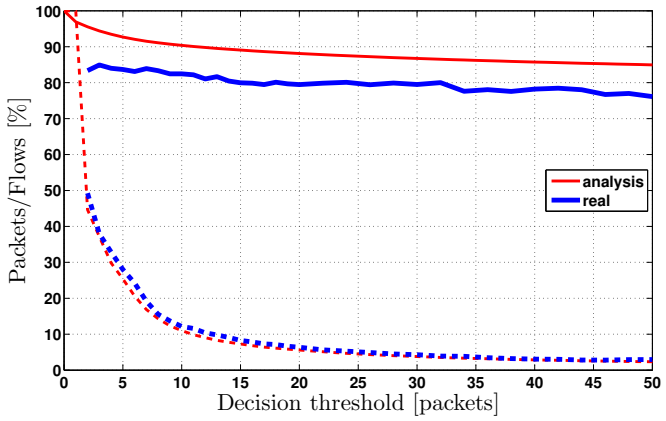

Fig. 11. Portions of offloadable packets and flows using the simple heavy flow detection method

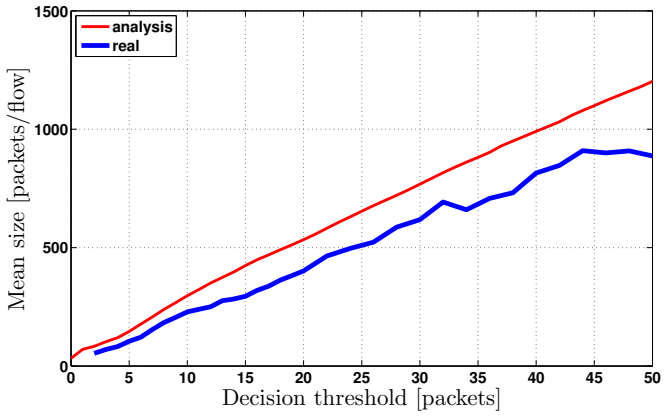

Fig. 12. Mean number of offloadable packets per flow in flows selected using the simple heavy flow detection method

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.

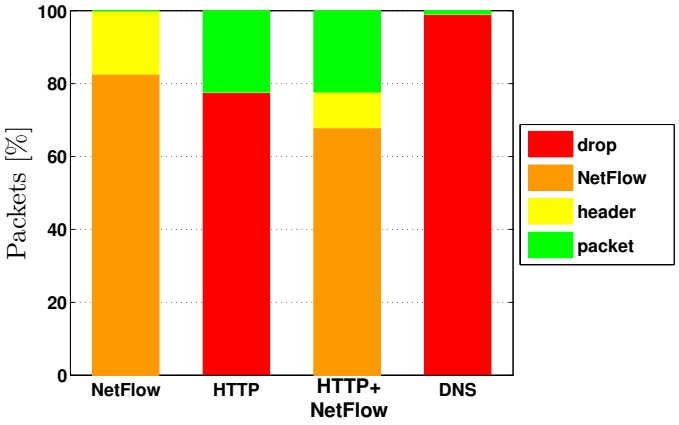

Fig. 13. Portions of hardware preprocessing types in tested use cases

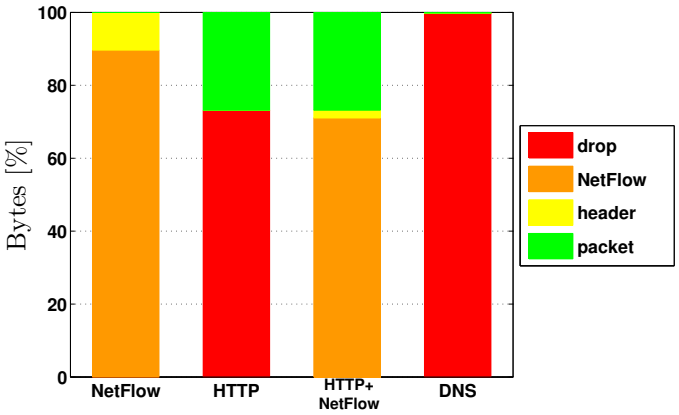

Fig. 14. Portions of hardware preprocessing types in tested use cases

	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.