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

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

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

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

## I. Introduction

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

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

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

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

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

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

## II. Analysis

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

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

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

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

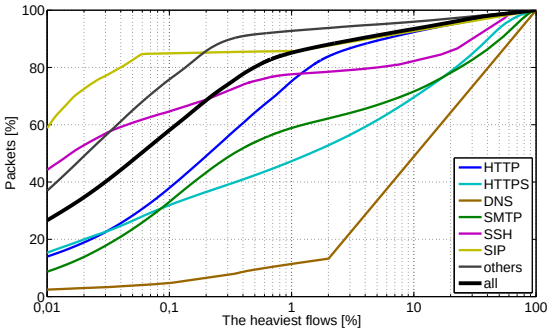


Fig. 1. Portions of packets carried by the percentage of the heaviest flows

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

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

## III. Architecture

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

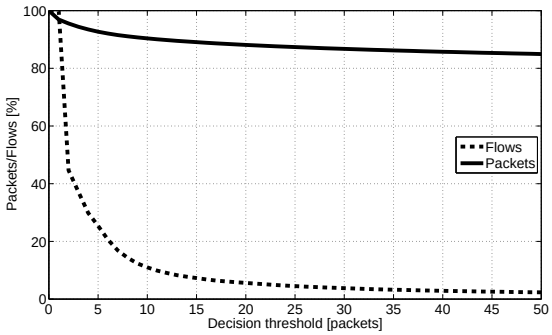


Fig. 2. Heavy flow detection using the simple method – portions of offloaded flows and packets

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

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

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

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

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

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

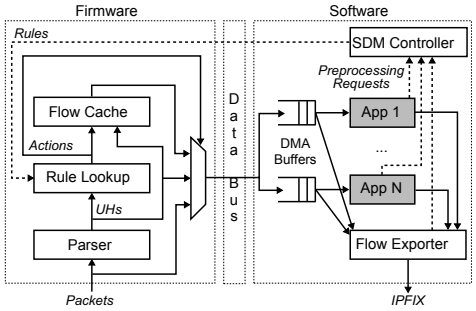


Fig. 3. Conceptual top-level scheme of SDM system

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

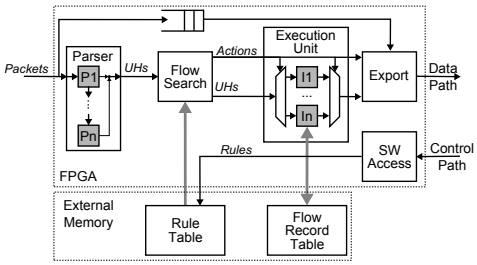


Fig. 4. Detailed firmware scheme

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

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

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

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

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

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

### *A. Execution Unit functionality*

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

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

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

## IV. Results

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

### A. Achieved performance

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

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

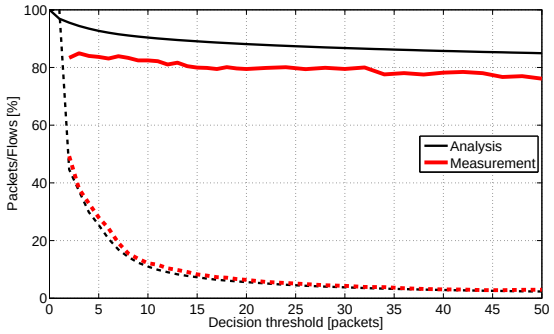


Fig. 5. Portions of offloadable packets and flows using the simple heavy flow detection method

### B. FPGA implementation results

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

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

TABLE I
Resources of the SDM firmware

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

TABLE II
Resources of the instruction blocks

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

## V. Conclusion

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

## Acknowledgement

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

## References

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

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

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

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

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

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

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