NETWORK PROBE FOR FLEXIBLE FLOW MONITORING

Martin Žádník, Jan Kořenek,

Faculty of Information Technology
Brno University of Technology
Brno, Czech Republic
email: {izadnik,korenek}@fit.vutbr.cz

Petr Kobierský and Ondřej Lengál

CESNET, z. s. p. o.
Zikova 4,
Prague, Czech Republic
email: {kobiersky,lengal}@liberouter.org

ABSTRACT

Research in measurement and monitoring of Internet traffic is evolving rapidly but high-speed network tools that would be able to follow it are still rare. New approaches and methods are often tested in offline environment or on low-speed links using software solutions, but consecutive real-time deployment on high-speed links is missing. In this context we propose a flexible network probe which is a foundation stone for further network measurement and monitoring. The architecture of the probe is based on a network acceleration card with Field-Programmable Gate Arrays (FPGA) and a host computer. The configuration for FPGA chips is automatically generated by a configuration program according to the user's definition of the monitored values in order to save hardware resources and increase the throughput. The definition of the monitoring process is described using XML, transformed to VHDL and synthesized. This enables the probe to gain any information about network traffic, assign it to the flow and process it, all of which can be arbitrarily defined by the user.

1. INTRODUCTION

At present, network traffic rates are continuously growing as well as the demands for effective network monitoring. Network administrators need tools capable of sophisticated, higher level semantic processing and analysis to cope with increasing network attacks and adversary activities. In recent years the flow monitoring technology usage has become widespread in the form of Cisco NetFlow and IPFIX.

NetFlow agents determine the number of bytes, packets, flag fields, duration of flow (defined as the sequence of packets with the same IP addresses, ports numbers and protocol). NetFlow technology helps to optimize the network infrastructure, reduce the operation costs and improve the capacity of planning and security incident detection.

The information gathered from NetFlow data can be used in higher-level applications, e. g. the detection of attacks using various methods [1, 2, 3] is very popular.

Flow based statistics can also be used for application protocol detection in the middle of the network, which is necessary for per-application traffic engineering, capacity planning, performance monitoring and security. Standard traffic identification methods associate the observed traffic with application according to the TCP or UDP port numbers. To avoid detection, several applications has begun to use dynamic port numbers and well-known ports commonly used for protocols, such as HTTP or SMTP. Using extended flow based statistics applications usually leave distinct fingerprints from which they can be identified.

Protocol identification techniques based on payload analysis [4, 5, 6] also exist, but they can be circumvented using variable length padding and protocol encryption. A better approach to protocol identification seems to be the behavioral traffic classification method [7, 8]. This method is able to identify applications with the accuracy of about 90 – 100 % using statistical information gathered from network flows. The method which combines both approaches uses payload data statistics from the first kilobyte of the traffic for very accurate protocol detection [9].

A number of methods have been implemented in the software and tested offline on small traffic samples or online on low speed networks. Their performance is not sufficient for deployment on current networks and dedicated accelerated implementations are required. Moreover, the lack of large representative testing data sets indicates the need for powerful tool to gather data from the network.

However, building new hardware for every application or design hardware which supports all presented applications is very costly. Since all enumerated studies process network traffic as variably defined flows, we suggest to accelerate general processing of flow, which remains the same for different applications.

In this paper, we propose a flexible hardware solution based on flows, which is optimized for specific monitoring requirements defined by the user. It allows to save valuable FPGA resources and increase the throughput. In addition, a framework for easy customization of monitored network characteristics is shown. We believe that this will encourage

researchers to evaluate their methods directly on real traffic.

The structure of the text is as follows: in section 2, we discuss the architecture of the probe for high-speed networks, section 3 describes the process of the configuration of the probe — the transformation of the user-defined XML file into the description of the hardware and the software. In section 4, the performance of the architecture is evaluated and section 5 briefly summarizes the study.

2. ARCHITECTURE

The proposed network probe is based on a commodity PC running Linux OS with a network acceleration card equiped with FPGA and memory. The architecture utilizes an available ten-gigabit card developed in Liberouter project [10]. The card provides a unified interface using the NetCOPE platform [11] to access its peripherals (network interfaces, memories, PCI bus), which allows fast architecture implementation.

The monitoring process is divided between the acceleration card and the host PC (see Fig. 1). This is a very different approach in comparison to previous architecture [12] of the flow monitoring probe on COMBO cards, where the monitoring process was implemented strictly on the card and the host PC only exported the received flow records. The idea of partitioning the process is supported by the fact that the host PC has enough processing power, which allows to move some tasks from the acceleration card to the PC, thus making the FPGA design simpler and faster.

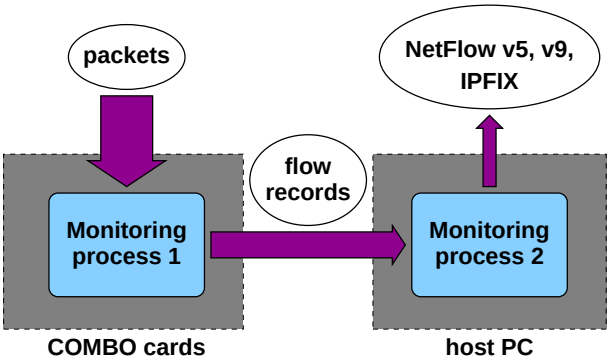

Fig. 1. Concept of the flexible FlowMon probe.

The two-stage monitoring process works as follows:

In the card:

- packets are received at the line rate,
- information is extracted from the packet,
- the flow key is hashed, which results in the direct address of the flow record in the records memory,
- collisions are solved by replacing the old flow record with the new one,
- expired or colliding flow records are transferred to the memory of the host PC.

In the PC:

- flow records are transferred using the busmaster DMA engine,
- another monitoring process aggregates flow records from the card into complete flow records,
- expired flow records are exported.

Such partitioning of the task allows to eliminate the number of fragmented flows, i. e. flows that were expired because of other reasons than timeouts (collisions or lack of memory). Furthermore, the analysis of several traffic samples has shown that the aggregation performed in the card can decelerate the incoming traffic speed to ten percent or less of the original value (depending on the size of the card memory). In this case, the processor is able to process up to ten gigabits of the original traffic. Closer details are given in section 4.

2.1. Hardware Architecture

The hardware design is based on two cores. The first one, NetCOPE core, provides an abstract layer to access hardware resources on the card, the other, FlowContext [13] core, is a management system intended for storage and load-balancing of context information among several processing units.

The FPGA configuration is composed of several units which are chained in the processing pipeline (see Fig. 2). Some parts of the processing pipeline are instantiated multiple times to overcome possible bottlenecks. The hardware design architecture can be divided into two logical parts: the packet parsing process and the metering process.

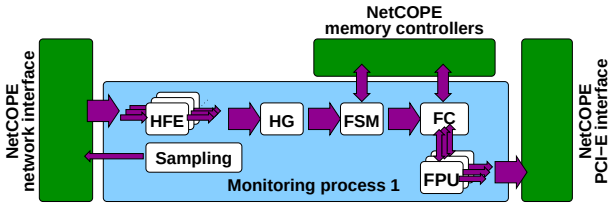

Fig. 2. Block diagram of the hardware design.

At first, packets with assigned timestamps are processed by several Header Field Extractors (HFE). HFE is a processing unit that extracts information from an arbitrary protocol packet header. The extracted information is used to create so called unified header record which contains the data for

the metering process. The HFE unit is implemented using Handel-C [14] and can be configured to extract all necessary header fields.

The fields that determine the flow (create the flow key) are the input of the hash function performed in the Hash Generator (HG), the result of which is the address to the flow records memory. Collisions caused by the hash function (two flows map to the same memory location) are detected by the comparison of all identifiers of the flow key during the update of the flow record in the Flow Processing Unit. If a collision happens, the old record is expired while the new record replaces it. Simulations show that a good hash function and sufficient memory capacity will keep the collisions at reasonably low rate.

To keep the states of all flows in the memory, the Flow State Manager (FSM) is used. The state of flow means the information about its lifetime. It allows to identify those flows which have already ended and can be released from the memory. The flow is considered to be finished after a certain time when no packet with the given flow key arrives. Therefore, FSM keeps track of the timestamp of the last seen packet of each flow and if the interval between the current time and the time of the last seen packet is greater than the inactive timeout (set by the user), then the flow is expired.

The core of the metering process is implemented in the Flow Processing Unit (FPU), which collects information about packets into flow records. The FPU is connected to the FlowContext (FC) interface, which is based on random memory access to any item of the flow record and any item of the unified header. The FC also allows to connect several FPUs and balance the load among them. The assignment of flow records to individual units must be atomic; this means that if one unit is processing a flow record, no other unit may work with the same record in parallel.

The design of the FPU is generated according to the definition provided in the XML file. The FC passes the packet data to the FPU together with the flow record and a command which instructs the FPU to perform certain operation. The unit also checks the flow identifiers for exact match in order to detect possible collisions of the hash function; colliding records are released to the host PC to be further processed by the software. The packet header and payload and the flow record then enter the update unit (see Fig. 4), which is described in section 3. The update unit aggregates the data in the flow record by the values in the header or the payload of the packet. If the flow record is empty (i. e. the currently processed packet is the first packet of the flow), the update functions need to use default values, as the values in the flow record are not valid.

When the update functions are computed, the result is checked using the control operations (if there are any defined). If the result is determined to be invalid, it is released to software where the processing continues.

2.2. Software Architecture

The operations of the flexible FlowMon probe can be divided into two logical phases - the preparation phase and the monitoring phase. The preparation phase covers all activities before running the probe for the first time. The user can specify their own requirements on the monitoring process and create a customized FPGA configuration. The monitoring phase includes downloading the configuration into the FPGA on the COMBO card, its initialization, configuration and network monitoring.

2.2.1. Network Monitoring (Monitoring Framework)

The control system of the probe is illustrated in Fig. 3. It consists of:

- the web frontend on a remote computer running a web server,
- configuration daemon on the probe,
- NETCONF system for communication.

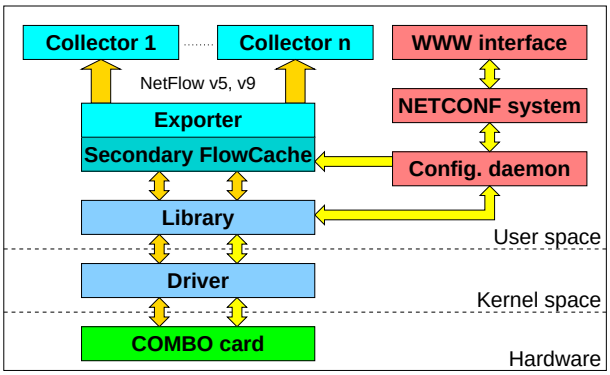

Fig. 3. Software layers — remote configuration.

The FPGA design is downloaded into the COMBO card remotely and the parameters of the probe (timeouts, samplings etc.) are set to their default values (startup configuration). The user can reconfigure any of the parameters of the probe including the specification of the export protocol and collectors for sending NetFlow records.

2.2.2. Secondary Flow Cache in the Software

We have developed a new method for the flow cache storage. Our solution is based on dividing the flow cache between the hardware and software, which enables to increase the capacity of the cache while retaining the full speed of the probe. Expired flow records are transferred from the flow cache on the COMBO card to the secondary flow cache in the software. The expiration is set by timeouts, collisions or flow cache capacity.

The secondary flow cache works with the flow records in the same way as the flow cache on the COMBO card with packet headers. The secondary flow cache significantly increases the limits of the probe in monitoring of high numbers of flows on high-speed networks.

3. PROBE CONFIGURATION

As mentioned above, the user configuration of the probe is provided in the form of XML structured data file. The XML format has been chosen as a human-legible well standardized means for data storage. The configuration file contains definitions of the *Header*, *Payload*, *Parameters*, *Flow* and *Controls* structures, which are:

Header Specifies the IPFIX names of header fields which are to be extracted from the received packet.

Payload Defines the ranges of payload data to be extracted from the packet.

Flow Specifies the output format of the flow record and the update functions of the probe. C-like expressions are used to describe the update operations of the flow record. The operands in the expressions may be either number literals or fields defined in the Header, Payload, and Flow structures.

Controls The definition of the control operations resides in this section, the purpose which is e. g. to avoid counter overflow by releasing the Flow record before the overflow may appear. The conditions for record release are described with C-like expressions similar to Flow definition; the operands are either number literals, Flow fields or user-defined thresholds. The control operations run after the flow record is updated according to the operations defined in the Flow.

The hardware design of the probe is tailor-made from the user-defined XML file by the core-generator program. This approach is necessary in order to achieve the required throughput of the probe with minimum resource consumption. The core-generator parses the input XML and constructs the data flow graph (see Fig. 4) of the processing chain, with the input being the Header and Payload of the incoming packet, Parameters (set by the software) and current Flow record (provided by the FlowContext) fields. The fields are connected to the ports of the components that perform the update operations, as defined in the Flow structure. The results of the update operations are then checked in the control component and stored in the updated Flow record.

The functions that can be used in the definition of the Flow and Controls are provided in a separate file with mapping onto hardware components. User functions may be

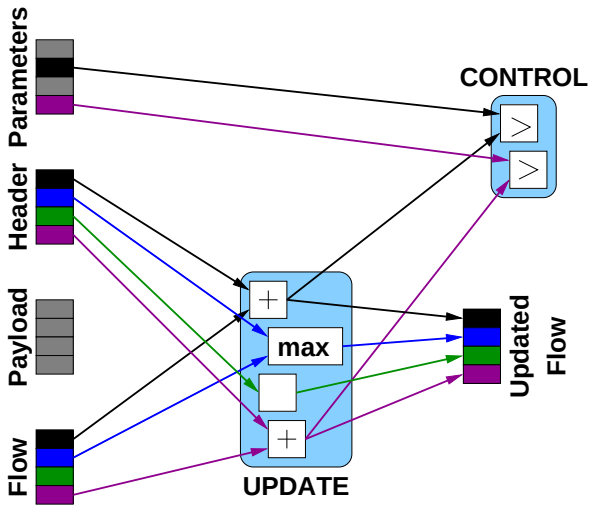

Fig. 4. Example of the data flow graph.

added for specific functionality and used as ordinary operations. The core-generator transforms the data flow graph into a set of VHDL files that specify the instantiation of the components and interconnection among them.

4. EVALUATION

To determine the throughput of the probe, bottlenecks of the suggested system have been analyzed. The system is based on the NetCOPE platform and the FlowContext system with the FPU, which collects statistical data. The NetCOPE platform does not affect the speed of hardware processing. The FlowContext system has a configurable data path width and on the grounds of the analysis in [13], it can operate at 10Gbps rates. Therefore, in order to determine the throughput, we need to focus mainly on the FPU.

The FPU is connected to the endpoint of the FlowContext system and when the packet processing is requested, it must conduct a set of operations, as illustrated in Fig. 5. At the beginning of processing, it is determined whether the item in the flow cache is valid or whether a new record is to be created. The new record creation consists of *(i)* the initialization of the statistical information based on the extracted packet header fields, and *(ii)* setting of the validity flag of the record. If the given flow record already resides in the flow cache, it is necessary to verify that no collision has appeared. In this phase, it is required that the identification fields extracted from the packet header are compared to the identification fields stored in the flow cache. If the fields differ, a hash function collision had appeared when searching the flow cache and the FPU sends a message to the FlowContext to release the original flow, and continues in the processing of the packet as if a new record is to be created. If no collision occurs, the statistical information of

the flow is updated according to the data from the packet header. Depending on the application, the update may work only with the header fields, or also with the payload.

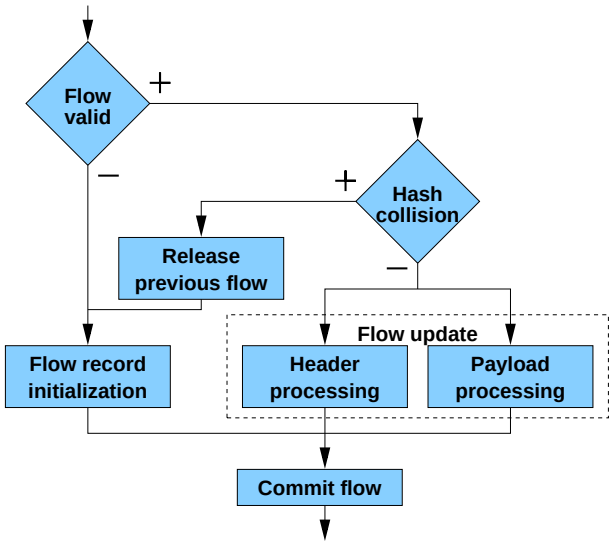

Fig. 5. Algorithm used by the FPU to process every packet.

The FlowContext provides the FPU with the access to statistical information and extracted fields through the memory interface. As the majority of fields in the packet headers are 16 bit-wide and the processing is optimized for this size, the proposed architecture uses the size of 16 bits for the memory access. The memory interface throughput is then the limiting factor of the FPU, because the speed of operations depends on the rate of data input. Hence two independent ports are used for the memory access, and the FPU throughput analysis is carried out from the point of view of the number of statistical data memory accesses.

The analysis of the FPU is based on the algorithm illustrated in Fig. 5 and the number of memory accesses in each processing step. To determine the validity of the record, it is necessary to check only one bit using a single memory access ($t_{valid} = 1$). The flow record initialization time, t_{init}, is linearly dependent on the record size, FR_{size}, and the time necessary for the collision detection is linearly dependent on the flow key size, ID_{size}. For the chosen 16-bit data width and two memory interfaces, it is possible to express the time for the initialization and collision detection by (1).

$$t_{init} = \frac{1}{4} FR_{size}, \quad t_{collision} = \frac{1}{4} ID_{size} \tag{1}$$

The update time of the flow record depends on a particular application. Some applications work only with the header data, while other can also use the packet payload. The analysis must take both cases into account and consider the maximum computation time, $\max(t_{header}, t_{payload})$. The computation time is constant for headers. We will consider the worst case, when the whole record except for the identification fields is updated and the size of the updated data is $FR_{size} - ID_{size}$ During the update, it is necessary to read out and write back the flow record, which results in twice as many memory accesses. The overall header update time for the data width of 16 bits and two interfaces is given in (2).

$$t_{header} = 2 \cdot \frac{1}{4}(FR_{size} - ID_{size}) = \frac{FR_{size} - ID_{size}}{2} \tag{2}$$

When working with the packet payload, the computation time is a function of the packet length, P_{length}, depending on the application. The most common payload processing operation is searching with linear worst case complexity; the processing time depends on the number of bytes, k, processed in a single clock cycle. If we take into account this case, the payload processing time can be expressed as

$$t_{payload} = \frac{1}{k} P_{length} \tag{3}$$

Proceeding from Fig. 5 and considering defined processing times, we are able to compute the worst case processing time, t_{worst}, in number of memory accesses from (4).

$$t_{worst} = t_{valid} + t_{collision} + \max(t_{header}, t_{payload}) \tag{4}$$

Using the model, throughput analysis of a single FPU has been carried out for *(i)* the case of packet header processing, and *(ii)* the case of packet payload processing algorithm. In the former case, the analysis has taken into account various flow key and flow record sizes. Fig. 6 displays the graph for the record widths of 32, 64 and 128 bytes. The analysis of the packet payload processing assumes that the payload processing algorithm complexity linearly depends on the packet length. The graph in Fig. 7 displays the throughput according to the packet length with various numbers of bytes processed in a single clock cycle.

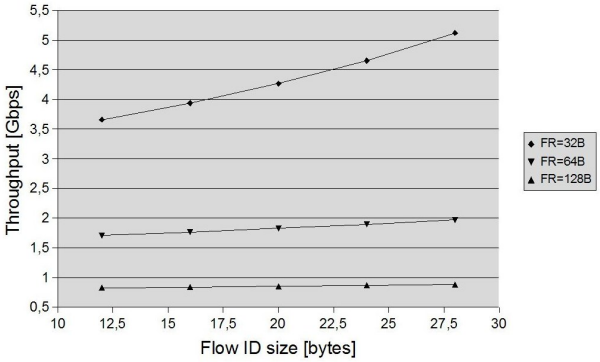

Fig. 6. FPU throughput for packet header processing.

As can be seen from both graphs, multiple FPU units are needed in order to achieve ten-gigabit throughput. The number of FPU depends primarily on the flow record size and payload processing speed. For the most common flow record size (64 bytes), seven FPUs and $k \geq 4$ are enough to process ten-gigabit line at wire speed.

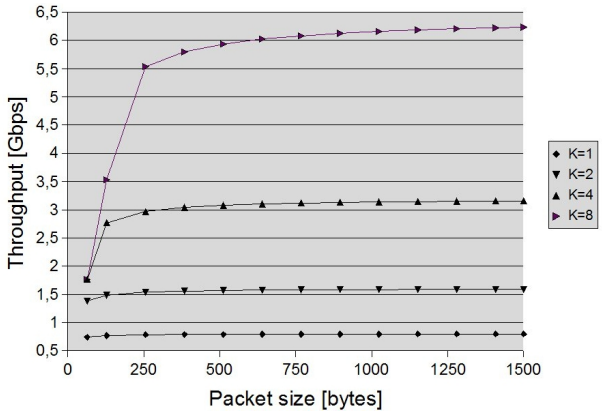

Fig. 7. FPU throughput for header and payload processing.

5. CONCLUSION

In this study we have proposed variable architecture that enables monitoring of various network traffic characteristics, which is required for full support of NetFlow v9 and IPFIX monitoring protocols. The versatility of configurations of the probe allows its usage even further, e. g. for application protocol identification or in an intrusion detection system.

The framework for the probe configuration has been proposed as well as the algorithm for generating the hardware design. The XML scheme allows extensions of the monitoring process, and the network administrator can easily customize it via web-frontend.

The architecture of the probe was analysed for performance and the FPU was identified as the bottleneck of the design with the conclusion that the throughput depends on the flow record size and payload processing speed. For the most common flow record size (64 bytes) and $k \geq 4$, seven FPUs are enough to process ten-gigabit line at wire speed.

Our future work will be focused on finishing the flexible FlowMon probe implementation, improving its performance by the usage of fine grain parallelism, and on extending its capabilities by adding extra features, such as an application decoder. After that, the probe should be deployed on real networks, where we are planning to test its capabilities. A challenging task would be to find or implement a collector that can understand all data exported by the proposed probe.

6. ACKNOWLEDGEMENT

This research was supported by the Research Plan No. MSM 6383917201 and Research Plan No. MSM 0021630528.

7. REFERENCES

[1] Y. Gu, A. McCallum, and D. Towsley, "Detecting anomalies in network traffic using maximum entropy," 2005. [Online]. Available: citeseer.ist.psu.edu/gu05detecting.html

[2] P. Barford, J. Kline, D. Plonka, and A. Ron, "A signal analysis of network traffic anomalies," 2002. [Online]. Available: citeseer.ist.psu.edu/barford02signal.html

[3] J. D. Brutlag, "Aberrant behavior detection in time series for network monitoring," in *LISA '00: Proceedings of the 14th USENIX conference on System administration*. Berkeley, CA, USA: USENIX Association, 2000, pp. 139–146.

[4] S. Sen, O. Spatscheck, and D. Wang, "Accurate, scalable in-network identification of p2p traffic using application signatures," in *WWW '04: Proceedings of the 13th international conference on World Wide Web*. New York, NY, USA: ACM, 2004, pp. 512–521.

[5] A. Moore and K. Papagiannaki, "Toward the Accurate Identification of Network Applications," in *Proceedings of the Passive and Active Measurement Workshop (PAM2005)*, March/Apri 2005, emeritus.

[6] H. Dreger, A. Feldmann, M. Mai, V. Paxson, and R. Sommer, "Dynamic application-layer protocol analysis for network intrusion detection," in *USENIX-SS'06: Proceedings of the 15th conference on USENIX Security Symposium*. Berkeley, CA, USA: USENIX Association, 2006, pp. 18–18.

[7] H. Jiang, A. W. Moore, Z. Ge, S. Jin, and J. Wang, "Lightweight Application Classification for Network Management," in *Proceedings of the SIGCOMM Workshop on Internet Network Management 2007: The Five-Nines Workshop*, August, 2007.

[8] J. P. Early, C. E. Brodley, and C. Rosenberg, "Behavioral Authentication of Server Flows," in *ACSAC '03: Proceedings of the 19th Annual Computer Security Applications Conference*. Washington, DC, USA: IEEE Computer Society, 2003, p. 46.

[9] P. Haffner, S. Sen, O. Spatscheck, and D. Wang, "ACAS: automated construction of application signatures," in *MineNet '05: Proceeding of the 2005 ACM SIGCOMM workshop on Mining network data*. New York, NY, USA: ACM, 2005, pp. 197–202.

[10] "Liberouter." [Online]. Available: http://www.liberouter.org

[11] T. Martínek and J. Tobola, "Interconnection System for the NetCOPE Platform," CESNET, Tech. Rep. 34, Dec. 2006. [Online]. Available: http://www.cesnet.cz/doc/techzpravy/2006/netcope-interconnection/

[12] P. Čeleda, M. Kováčik, T. Koníř, V. Krmíček, P. Špringl, and M. Žádník, "FlowMon Probe," CESNET, Tech. Rep. 31, Dec. 2006. [Online]. Available: http://www.cesnet.cz/doc/techzpravy/2006/flowmon-probe/

[13] M. Košek and J. Kořenek, "FlowContext: Flexible Platform for Multigigabit Stateful Packet Processing," in *2007 International Conference on Field Programmable Logic and Applications*. IEEE Computer Society, 2007, pp. 804–807.

[14] T. Dedek, T. Marek, and T. Martínek, "High Level Abstraction Language as an Alternative to Embeded Processors for Internet Packet Processing in FPGA," in *2007 International Conference on Field Programmable Logic and Applications*. IEEE Computer Society, 2007, pp. 648–651.