

Flexible FlowMon

Martin Žádňík, Petr Špringl, Pavel Čeleda

22.10.2007

1 Abstract

Research in measurement and monitoring of Internet traffic is evolving rapidly but dedicated tools that would be able to follow it are rare. New approaches and methods are often tested in, so called, offline environment using software solutions, which is of course necessary for effective development, but consecutive deployment of hardware tools on the high-speed networks are missing. For a lot of cases, we hope that our new concept of Flexible FlowMon probe can fill-in the gap between off-line and on-line measurement. Main objective of Flexible FlowMon probe is to gain any information about network traffic and assign it to the flow which can be arbitrarily defined by user. The architecture of the probe is based on COMBO cards and host computer. The firmware of the card is completely new and we believe more scalable than previous implementations of our NetFlow probes.

Keywords: Monitoring, FlowMon, NetFlow, security

2 Introduction

Recent development in measurement and monitoring of network traffic is driven by increasing demands on quality and security. Quality of the network connections is vital for emerging real-time applications such as video and audio telecommunication, streaming, etc. Since quality measurement has a long history, security on the Internet is phenomena of last decade. With growing number of Internet users the interest to abuse Internet for the illegal activities also grows. According to several studies the Internet crime has become more organized and is carried out for money. Nevertheless, both areas (quality and security) are subject of intensive research and new techniques are developed to measure and monitor traffic more accurately. Over last decade, several methods to detect on-going attacks, spreading worms and other malicious traffic or events were proposed.

In paper [GMT05] the method for detection of anomalies in network traffic using maximum entropy was proposed. The method was validated on packet traces

captured by DAG cards¹. Only IP addresses and ports were necessary to construct the state space upon which the algorithm looks for anomalies. Another work [KSV07], focused on scalable attack detection, suggests to aggregate traffic more than per flow thus saving more space. This approach utilize Partial Completion Filters (PCF) which can be modeled as several flow caches each with different hash function. It would be interesting to alter the flow probe to perform such measurement. A pioneering work to statistically characterize network traffic was presented in [Pa94]. And lot of others carried on, for instance [Tru05]. Several works, e.g., [Cro07], [McG04], [MZ05], were focused to classify traffic according to applications utilizing statistical information. Some [DWF03], [MP05] were focused even more to precisely identify certain applications. Besides new methods also an architecture of software network monitor was proposed in [Moo03]. It distributes data to several components in order to capture data at different levels, network, transport and application. Such scheme seems to be complete enough but may quickly exhaust most of the computational power. Therefore the authors suggest to balance traffic among several PCs.

Most of the papers share similar characteristics:

- Processing at the level of flows
- Offline processing
- Small traffic samples or low rate traffic samples

Published methods were implemented in software and tested offline on small traffic samples. Performance of software tools is not sufficient for deployment on current networks and dedicated accelerated implementations are required. Moreover the lack of large testing data indicates the need for powerful tool to gather data from network. It seems that a lot of methods process network traffic at the level of flows, no matter of what level of flow is considered and no matter of what information is necessary. See [Qui04] for closer details about flow monitoring.

Therefore a flexible hardware solution (that would minimize effort for its customization) based on flows would do most of the work while the rest of specific processing could remain implemented in software. Proposed flexible FlowMon probe fulfill these requirements and we believe that it will encourage researchers to evaluate their methods directly on real traffic.

¹<http://www.endace.com>

3 NetFlow Background

During last three years Liberouter team developed two accelerated NetFlow probes. First one was a proof of concept that shows feasibility to implement IP packet flow-monitoring on board with FPGA chips. Despite its performance was poor, it outperformed standard software probes. The second implementation was improved in both functionality and performance. Currently it is able to monitor up to 3 million packets per second (holds for shortest packets) or up to five gigabits (holds for longest packets). The advantage of autonomous NetFlow probe is the possibility to deploy NetFlow measurement where a dedicated source of NetFlow data is missing. Autonomous NetFlow probes allows user to set any parameter arbitrarily without fear that the bypassing traffic will be influenced. On the other hand when enabling NetFlow on routers one have to always be careful about the additional performance demands on hardware resources by monitoring process. Probes are usually more flexible and allow to test new features such as different types of sampling, filtering or anonymization.

Probes were successfully deployed in CESNET network and also in networks of GEANT2 partners (SWITCH, SURFnet, GRNET). Thanks to the testing and other activities, Liberouter members are involved in, our experience with flow measurement has grown. We realized that it is necessary to completely redesign the concept of the probes and to add more flexibility.

The first drawback of current probes is limited performance due to software-like design of the firmware. If the flow monitoring is analyzed in closer detail several critical tasks can be discovered. These are packet-header parsing, addressing, keeping state of the flow and updating record of the flow (flow record).

Packet-header parsing is performed by small processors with parsing program for extracting relevant data (IP addresses, transport ports, protocols, length of packets, etc) which create or update the flow record. The processor is described in VHDL and synthesized in FPGA with rest of the firmware. It is well known that implementation of processor consumes more chip resources than dedicated component performing the same task. Waste of resources causes issues when scaling the design for higher data rates.

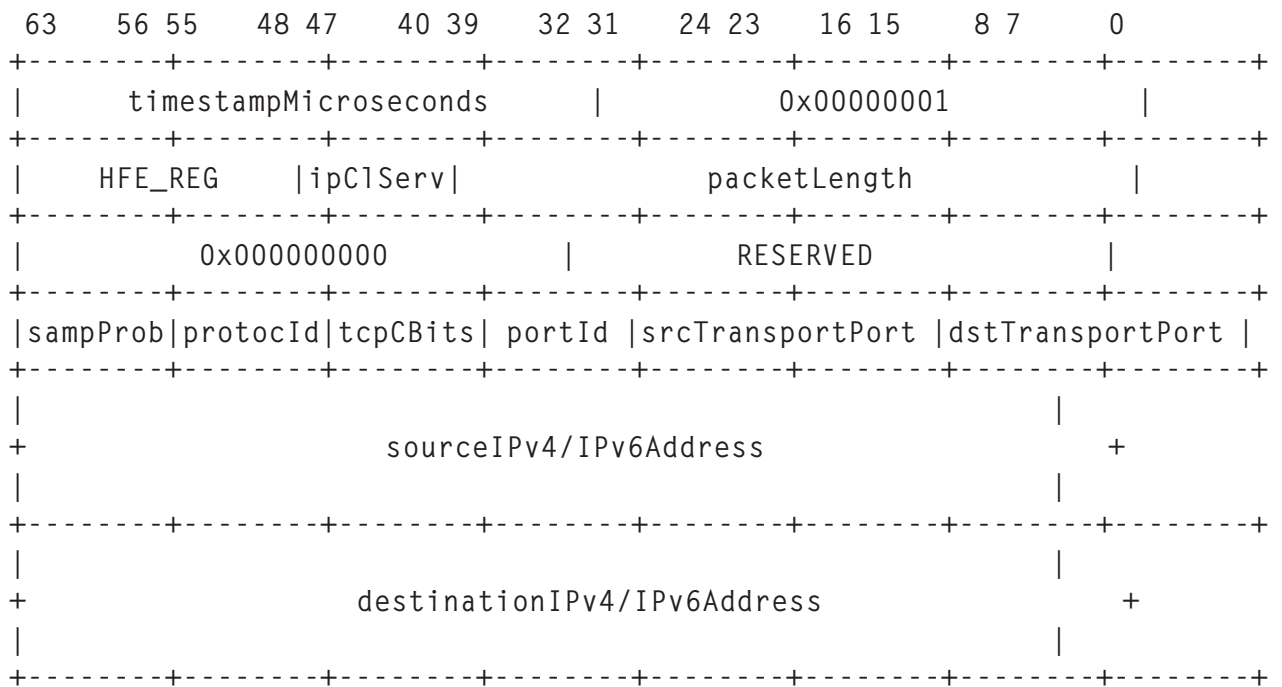
The addressing of the flow record is done by computation of hash on the flow key (IP addresses, ports, interface ID, protocol). The result of hash function is used to index a vector of pointers to the records of the flow cache, as suggested in [Mol04]. This is clearly an efficient way how to search and keep flow records in software implementation but for hardware it requires either two accesses in one memory or one access in two memories and both options are costly.

Another software-like solution is utilized to keep state of the flow records (it is necessary to release old flow records). The state is kept in an extra memory

where the timestamps of last arrived packet per flow are bounded doubled list. Every time a flow record is updated/created its timestamp is moved/created at the beginning of the list. The oldest timestamps remain at the end of the list and are precisely and easily identified. But the price paid for an extra memory to store list pointers is too high.

Finally, updates of flow records are not executed in parallel. It means that if the update of the flow record requires more complex operation which takes longer time, no other flow record is updated during this period.

The second drawback is caused by fixed definition of the flow record. User is restricted in what information can be extracted out of the packet header, how it is entered in the flow record, and how the flow record looks like. An example of fixed structure extracted from packet-headers is shown in the following diagram:



In order to modify flow record in current implementation, user would have to rewrite the program for parsing processor, define a mask for the flow key, and write a new unit for updating a flow record. These obstacles discourage users to experiment with the probe's abilities and prevent them to utilize all its potentials. Current definition of flow record is not sufficient in respect to introduction of new flow information export protocols, such as NetFlow v9 [Cla04] or IPFIX [Cla07]. These protocols allows to export whole variety of packet information and thus the monitoring device must be able to modify the definition of flow record. To our knowledge there are already new items of flow records that user would appreciate but are not defined in NetFlow v9 nor IPFIX. For example

several bytes of payload of first packet in the flow, time statistics about periods between packets of the same flow and others.

4 Beyond NetFlow

NetFlow is traditionally used for routing optimization, application troubleshooting, traffic mix monitoring, accounting and billing, anomaly tracking and others. Besides these running-up applications new utilization attracts the attention. Among the most widely known belongs detection of DoS attacks, already embedded in some collectors. The simplest heuristic for detection of DoS is based on deviation of number of flows from average number of flows for given period. This method is able to identify obvious DoS or scanning activities. Various correlation or search methods were proposed for closer insight in malicious traffic. These are usually based on search through IP address and port space which is constructed out of NetFlow data.

Previous applications operates with ordinary NetFlow data which works fine up to transport layer of TCP/IP model. But there are new applications that require flow-like data with more or different information reported. For some of them standard items of flexible NetFlow v9 or IPFIX would suffice. The rest is left with no other choice than to acquire traffic by an arbitrary means (e.g., tcpdump) and to extract its proprietary information on their own.

First of such application is network application decoder. Its function is to identify applications that are used in network. Some of them are naturally identified by its transport port number but a larger group either utilize unprivileged ports dynamically or hide its traffic on ports assigned to other applications. Ordinary NetFlow data can report only applications running on well-known ports with consequence of joining legitimate with malicious traffic on the same port.

The task of the decoder is to distinguish between various applications by inspecting the data at application layer (payload of transport layer) where it may look for specific patterns using regular expressions (e.g., for SSH connection: `^ssh-[12]\.[0-9]`). Such functionality is already supported in Linux netfilter framework [?], which uses open-source collection of application signatures of L7-filter². The L7-filter with modified signatures were also used and tested in customized architecture of NIDS [Dre06] with good results. The implementation of the network application decoder directly in the flow probe would be only a natural extension since it was the original intent to provide information about application traffic mix. Moreover the core of decoder is based on well studied problem of pattern matching which was shown to be suitable for hardware implementation [Dh04], [TBS06], [CS04], [KK07a].

²<http://l7-filter.sourceforge.net>

Besides pattern matching an alternative approach, statistical network behavioral analysis, was introduced to detect certain types of applications. The analysis is based on specific behavior of various application traffic rather than on pattern detection which is sometimes impossible concerning encrypted connections or simply applications where no significant pattern signature can be identified. Main motivation for statistical analysis is its robustness. For example, if the analysis is focused to detect interactive communication by monitoring interval between consequent packets it would be very hard for application to deceive detection mechanism by generating packets with larger interval between consequent packets as the quality of communication would suffer. Another advantage is that the learning phase (during which statistical indicators are chosen and thresholds are set) of the analysis can be automatized, for example [McG04].

Usually, indicators about packet length or interval between packets of the same flow are used and therefore the flow probe should support various aggregation schemes such as minimal/maximal, average, variance or construction of undersampled histogram. The analysis itself does not have to be performed in the probe. On the contrary, it would be better to have it on the collector where the analysis can leverage other indicators such as number of incoming/outgoing connections per host, etc. This way a potentially malicious traffic can be discovered (p2p file-sharing, worm spreading, etc.).

Measurement of connections quality is yet another application that could benefit from flow monitoring, especially where multimedia application are utilized. The advantage is that such measurement is non-invasive (no extra packet have to be launched in the network) and at the same time performed upon multiple hosts and applications during real traffic utilization. Gathered data can be correlated afterward which can help with troubleshooting and optimizations. Measurement of quality requires precise source of timestamps to be built in the flow probe and enough bits allocated for the timestamps in the flow record. Again additional operations to express distribution of interval between packets of the same flow are required. Besides standard flow items, the flow record must be amended with Type of Service (IPv4) or Class of Service (IPv6) field.

Our experience with monitoring using FlowMon probes shows that flow probes should also support monitoring of MPLS labels, VLAN tags and MAC addresses arbitrarily, depending on the deployment in the network. Another experience is that nearly any item in the flow record can be a keyfield, for example, if the keyfield is the length of packet then the result in the probe memory is precise histogram of packet-lengths observed in the network. In summary the flow monitoring process must be flexible in many directions but at the same time manageable.

5 Flexible Flow Monitoring

Inspired by different utilization of flow data we decided to create an XML schema that allows user to define:

- what is extracted out of the packet
- items of the flow key
- structure of the flow record
- how the flow record is updated by extracted information

The schema allows to define primarily structure of the flow record including placement and size of individual items but it also defines structure of so called UH-header (record with extracted items from the packet header). Individual items of both records are operands of assigned aggregate operation. Moreover each item of the flow record can be guarded by so called control operation which can trigger an arbitrary event, e.g., saturation of the counter of packet-length causes the flow record to expire.

The schema is divided into two files, definition of operations and definition of records. First file contains description and implementation of update and control operations (tree on the right side of Figure 1). Each operation has its unique name which is used to refer to the operation. Operation can be implemented in arbitrary number of languages. But VHDL and C implementation are points of our interest because our group use them to implement flow monitoring process in firmware or in software. It is supposed that the list of currently supported operations will be extended during deployment of our probes according to feedback of users. Extension is very easy and requires only to write short section of program implementing the update operation, the risk of introducing further errors is minimized. An example in VHDL of update operation (accumulation) is given:

```
-- input of the adder
data <= CONTEXT_IN when (FIRST = '0') else
    -- initialization accumulator when first packet arrives
    std_logic_vector(to_unsigned(DEFAULT_VALUE, CONTEXT_WIDTH));

-- sum of accumulator and input data
CONTEXT_OUT <= std_logic_vector(unsigned(data) + unsigned(DATA_IN));
```

Second file contains definitions of UH-record, payload and flow record which matches chronologically to extracting information out of packet-header, payload

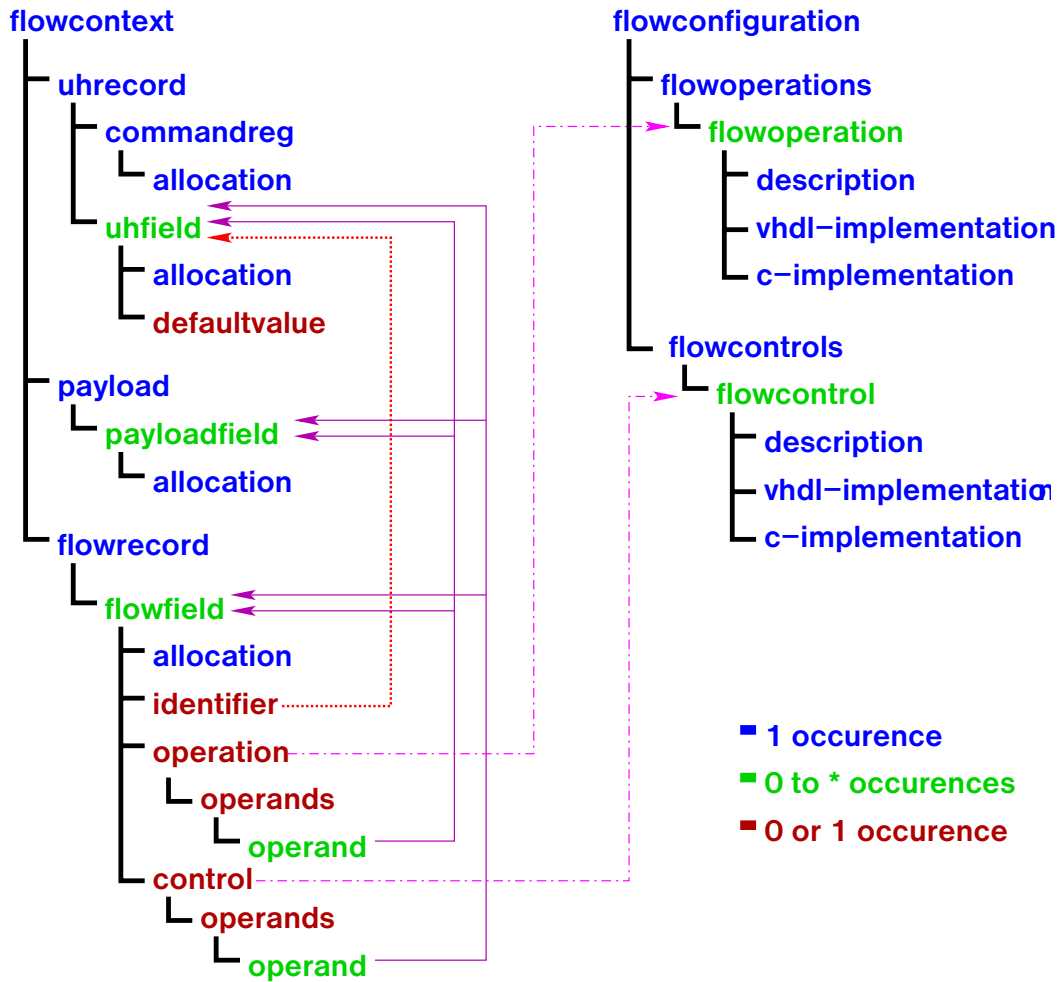


Figure 1: Schema of XML tree for definition of flow record processing

and then updating the flow record. First the UH-record defines which fields of packet-header are extracted, where is their position in the UH-record and how many bytes are allocated for their storage. It might seem strange to allocate sizes for packet-header fields whose size is already defined in respective RFC, but some items might have arbitrary size, such as timestamps, interface ID or user does not have to be necessarily interested in whole field. Each field contains a parameter whether it contributes to flow key or not. If the packet header does not contain required field default value is set instead. Following example illustrates the definition of source transport port:

```
<uhfield name="uh_sourceTransportPort" identifier="true">
  <allocation address="0x1C" size="2"/>
  <defaultvalue value="0x0000"/>
</uhfield>
```

The payload section describes what part of the packet payload is extracted. This section is not utilized for now but we expect its importance to grow, because of its utilization with application decoder.

All sections are connected through definition of flow record. Each item of the flow record (called flow field) refer to one update operation and one or none control operation. Operands of the operation can be from UH-record, payload and flow record sections and their reference is mapped as the input of the operation. Moreover if the flow field is marked as identifier than referred field in UH-record is checked if it contains the same value. Such operation might seem redundant but particular implementations of the monitoring process might take advantage of it. An example of flow field definition is given below. Note that the naming convention was adopted from definition of IPFIX wherever possible.

```
<flow field name="octetTotalCount">
  <allocation address="0xB" size="5"/>
  <operation name="accumulate">
    <generics>
      <generic id="default_value" value="0"/>
    </generics>
    <operands>
      <operand id="source" field="uh_ipTotalLength"/>
    </operands>
  </operation>
  <control name="control_overflow">
    <generics>
      <generic id="constant" value="0xFFFFFFFF00"/>
    </generics>
```

```
</control>  
</flow field>
```

Figure 1 and description of the XML schema was simplified for purpose of readability. Curious reader might notice that there is not a clearly defined way how to specify validity of items in any of the records (UH-record or flow record). A good example is monitoring of ICMP packets. If an ICMP packet is received, transport ports cannot be extracted and the flow fields containing these ports are not valid. For this purpose there is a extra bitmap field in each record that allows to mark each field valid or not.

It is supposed that the end-users of the probe do not need to care about XML schema at all. Instead, an implementation of web front-end is proposed that allows to select monitored items and assign operation from a predefined list. Such user interface would also allow to optimize the structure of the flow record according to the particular implementation of the monitoring process. The framework suited for our FlowMon probes is described in chapter Software.

6 Probe Architecture

The probe is based on commodity PC running Linux OS with network acceleration card. These acceleration cards are traditionally developed in LiberoRouter project because there are no other cards available that could provide ten gigabit network interfaces and at the same time a programmable chip (FPGA) that can process high data rates. Moreover each card provides a unified interface using a NetCOPE platform [MT06] to access its peripherals (network interfaces, memories, PCI bus) which allows to implement the architecture faster with potentially less errors. Despite unified interface each card is equipped with different peripherals. For purpose of flow monitoring new family of COMBO V2 card seems to be the best option. It provides enough bandwidth to transfer data from interface card to mother card via RockeIOs, contains faster Virtex-5 chips, and supports PCI-Express 8x.

The monitoring process is divided between acceleration card and host PC (see Figure 2). This is very different approach in comparison to previous implementations of FlowMon probe on COMBO cards where the monitoring process was implemented strictly on the card and the host PC only exported received flow records. The idea of partitioning the monitoring process is supported by our experience gained during performance testing of previous probes. We noticed that while the card was fully utilized, processor in the computer was hardly utilized at five percent of its time. Therefore we can take advantage of additional processing resource in PC and make the firmware simpler and faster.

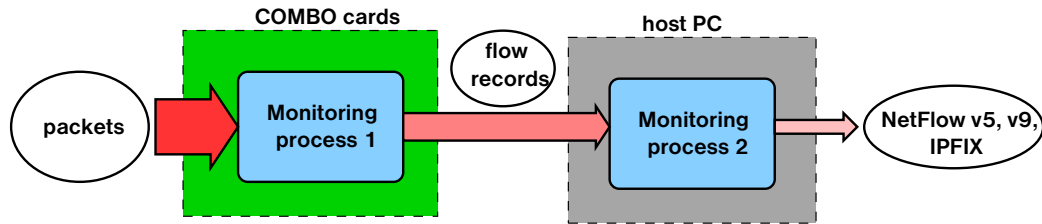


Figure 2: Concept of flexible FlowMon probe

The two stage monitoring process works as follows: In the card

- Packets are received at line rate
- Information extracted from packet
- Flow key is hashed and the result is direct address of the flow record
- Collisions are solved by replacement of old flow record by new one
- Expired or collision flow records are transferred to memory of host PC

In PC

- flow records are transferred using bus master DMA engine
- Another monitoring process aggregates records exported by firmware into complete flow records
- Expired flow records are exported

Such partitioning of the task allows to eliminate number of fragmented flows, i.e., flows that were expired because of other reasons than timeouts (collisions or lack of memory). Further, it was estimated by analysis of several traffic samples that the aggregation performed in the card can decelerate incoming traffic speed to ten percent or less of the original value (in dependence on the size of the memory on the card). In that case, the processor would be able to process up to ten gigabit of the original traffic. Closer details are given in chapter Performance Analysis.

7 Firmware Architecture

The firmware is based on two cores. NetCOPE core provides an abstract layer to access hardware resources on the card. FlowContext core is a management

system intended for storage and load-balancing of context information among several processing units. The utilization of these core by FlowMon firmware shown in Figure 3.

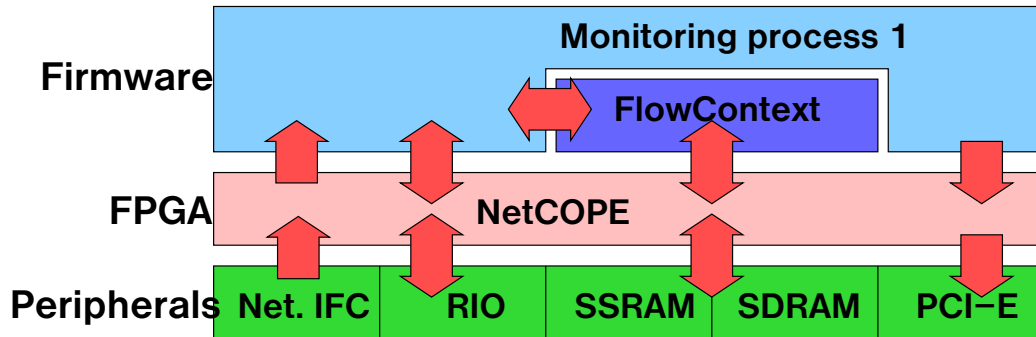


Figure 3: Concept of hardware and firmware architecture

The application firmware is composed of several units which are chained in processing pipeline (see Figure 4). Testing on real traffic and using Spirent packet generator identified bottlenecks of previous designs. Therefore some parts of the processing pipeline are instantiated multiple times. The firmware architecture can be divided into two logical parts, packet parsing process and metering process. Packet parsing chain is described first and then the description is focused on the metering process.

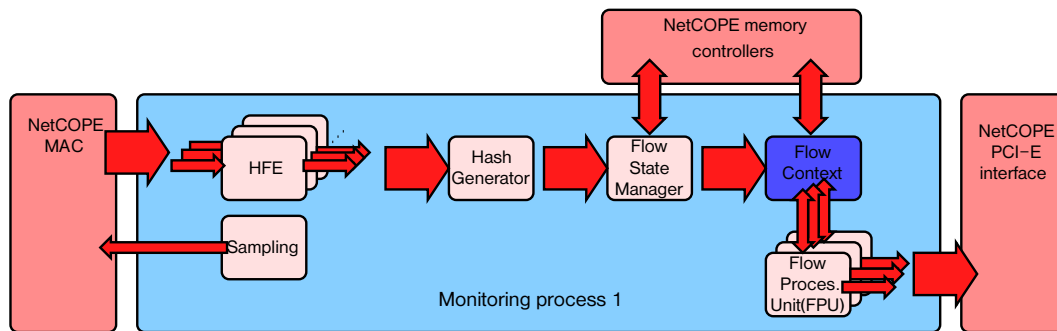


Figure 4: Block diagram of firmware

The application firmware utilizes NetCOPE network interface to receive packets. The interface delivers packets at Layer 2 of TCP/IP model to the firmware which means that received packets are already checked for the correct Cyclic Redundancy Check (CRC), correct Start Frame Delimiter (SFD) and minimum and/or maximum transfer length. The NetCOPE provides interface to sample received packets and to assign them unique timestamps. The sampling function is optional and is implemented in firmware as well as generating of timestamps can be either implemented in firmware or received from external GPS unit.

Packets with assigned timestamps are processed by several Header Field Extractors (HFE). The task of HFE is to extract information from the packet header. HFE uses extracted information to create so called unified header record which contains data for the metering process. The HFE unit is implemented using Handel-C which is a modification of standard C language specialized for description of parallel computing structures. HFE can extract nearly any field from headers of packet which implicates large consumption of computational resources. Therefore its complexity is reduced during preprocessing by definitions of monitored header-fields in the config.h file which is generated according to the XML configuration.

```
/* Source TransportPort */
#define uh_sourceTransportPort 1          /* Extract sourceTransportPort */
#define uh_sourceTransportPort_UH_ADDR 0x1A /* Address in the UH-record */
#define uh_sourceTransportPort_UH_SIZE 0x02 /* Size of the field */
```

Header Field Extractor, written in HANDEL-C [DMM07], has several advantages in comparison to previous HFE implemented as processor with assembler program. For example, its implementation consumes nearly same amount of resources as original HFE but it is able to process one and half million packets per second. Moreover the processing does not have to be stopped during jump commands, one word of data is processed each clock cycle. Therefore its performance is stable and predictable in advance. Ten HFE units suffice for processing of all packets at fully utilized ten gigabit link. The only drawback is that the the HFE cannot change its functionality by reloading a new program during execution. Instead, configuration file of the whole chip must be loaded because HFE is integral part of the firmware. It means that all previously monitored data are lost. Another issue that would emerge anyway is reordering of packets because of parallel execution on several units with different execution times. The correct ordering must be remained so it does not cause race conditions during updates of flow record. Therefore all packets are marked with sequence numbers before they are dispatched to HFE units. HFE units assemble unified headers which are ordered again by the sequence number into one stream together with packets payloads. These frames are transferred via RocketIO (RIO) to the mother card where the processing continues.

Fields that determine the flow are subject of the hash function. Its result is the address to the memory of flow records. Collisions caused by hash (two different flows mapped to the same memory location) are detected by comparing all identifiers of the flow key during update of the flow record in Flow Processing Unit. If the collision happens the old record is expired while the new record replaces it. Simulations show that good hash function and sufficient memory capacity will keep the collision rate reasonably low (see chapter Performance analysis).

Flow State Manager is intended for keeping states of all flows in the memory. State of the flow means an information about its lifetime. It allows to identify those flows which have already ended and can be released out of the memory. The flow is considered to be finished after certain time when no packet comes for given flow. Therefore the Flow State Manager keeps track of the timestamp of the last seen packet of each flow and if the interval between current time and the last seen packet is greater than the inactive timeout (parameter set by an user) then the flow is expired. Several possible algorithms are able solve this task. Please note that bit-length of time information can vary in following algorithms and it is independent on the timestamp assigned to the packet at the input interface.

Algorithm implemented in previous probes is based on the ordering of flow states according to time of the last seen packet. The quickest way how to do it is to keep states of flows in bidirectional bounded list. Each item has a timestamp and two pointers. The idea of ordering is simple. New or updated flows are always rebounded to the top of the list and their timestamps are updated. This way the least recently used (inactive) ones remain at the tail. It is easy to identify inactive flows by comparing timestamp of the last item in the list.

The second algorithm can be in short described as a field of sequentially decremented counters. It works as follows:

- Every incoming packet causes setting the counter for the given flow to the maximal value.
- All non-empty counters are periodically checked and decremented.
- If the value of counter reaches one then the flow is considered to be expired.
- After the flow is removed zero value is set into the given counter which signals that the item is empty.

The inactive timeout is changed by adjusting the speed of the periodic count-down of counters.

The third algorithm stores timestamp (generated in the Flow State Manager) of last seen packet for each flow in the memory. All valid items in the memory are again periodically checked whether the interval between the last seen packet is longer than the inactive timeout. Only significant bits of the timestamp are stored which leads to timestamp with limited range and low precision. Moreover if the inactive timeout is changed then all stored timestamps must be rescaled so that they fit into the range of current timeout. To remove this drawback we suggest to implement the counter generating timestamps with the same bit-length as the

memory word. The inactive timeout only increases/decreases the clock rate for the generating counter. The shorter the inactive timeout the faster the clock cycle and the sooner the flow is expired. This algorithm was chosen for its low memory and chip resource consumption and heftiness of implementation.

The core of the metering process is implemented in the Flow Processing Unit which aggregates information about packets into flow records. It is connected to the FlowContext which is described in [KK07]. The interface of FlowContext is based on random memory access to any item of the flow record and any item of the unified header. The FlowContext also allows to connect several Flow Processing Units and balance the load among them. The assignment of flow records to individual units must be atomic. It means that if one unit is processing the flow record then no other unit may work with the same flow record in parallel.

The design of the Flow Processing Unit is generated according to the definition provided in the XML file. The operation of the FPU is basically divided into three steps: loading, processing and storing. The loading and storing parts are straightforward, the data are either loaded from the FlowContext memory interface into a register array (loading), or the result of the operation is stored from a register array into the FlowContext memory (storing). However, the processing part is more interesting. First of all, the data bits containing the information about the command that is to be executed are extracted from the UH record and from the flow record, so that the operation of the FPU could be determined. If the update operation is requested, the identifiers fields of the UH record and the flow record are then checked for exact match. If the identifiers do not match, the flow record is released to software for further processing. Otherwise, the fields of the UH record, flow record and packet payload are processed in the aggregation unit, the design of which is generated from the file with definition of operations. The data are processed in parallel. The complexity of the aggregation operation is limited to basic fixed-point operations, such as addition, subtraction and multiplication, and bit operations as AND, OR, XOR etc. Nevertheless, it is possible to use nested operations (e.g., accumulate squares of differences of two values for the computation of standard deviation) or to implement additional operations. It should be noted that if the flow record is empty (i.e., the processed packet is the first packet of the flow), the aggregation functions need to use default values, as the values in the flow record are not valid.

When the aggregation functions are computed, the result is checked using the control operations (if any). If the result is determined to be invalid, it is released to software where the processing continues. Future improvements include pipelined processing, which will speed up the update operation and also enable sharing of the components used for the aggregation functions' computation.

Another feature to be implemented is an interface to the application specific engine, e.g., application decoder.

8 Software

The operations with flexible FlowMon probe can be divided into two logical phases - preparing phase and monitoring phase. The preparing phase covers all activities before running the probe for first time. User can specify own requirements on monitoring process and create a customized firmware. The monitoring part includes firmware booting into FPGA of COMBO cards, its initialization, configuration and network monitoring.

The software architecture follows partitioning into two phases. It contains the framework for creating (generating) firmware and the framework for probe starting up and management.

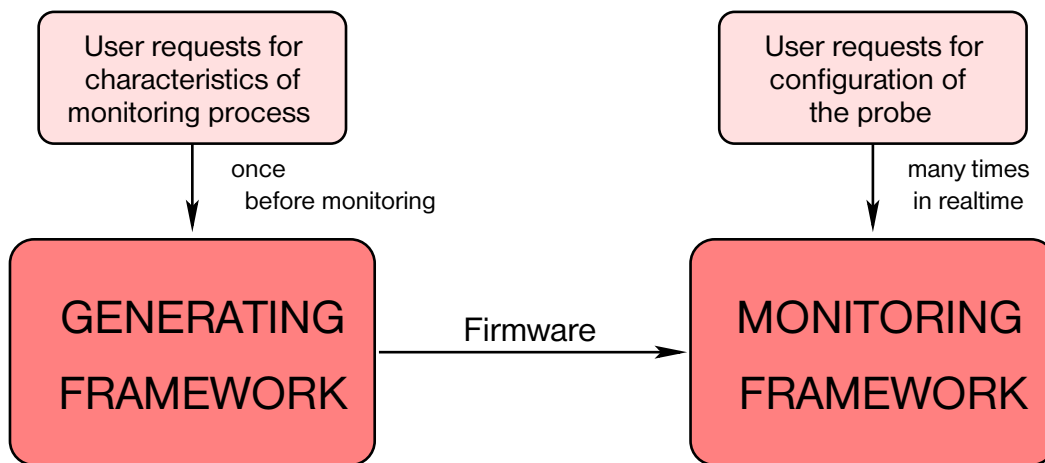


Figure 5: Software architecture

8.1 Phase 1 - Preparation for Monitoring (Generating Framework)

The framework consists of tools for describing monitoring process and firmware synthesis. It is presented like one tool with a web interface where user can specify the requirements on monitoring process. The consistency of user requirements is verified and a XML file created. The file unambiguously describes structure of flow record, unified-header record and operations with them.

Created XML file is processed and proprietary configuration files intended for synthesis are generated. Consequent synthesis of firmware may take more than

hour. Moreover, the synthesis process requires specific tools which are not publicly available. Therefore the synthesis takes place on dedicated server.

After the synthesis ends, user is informed by an email how to download a generated package. The package contains COMBO card firmware, XML file describing address space of firmware and XML file describing variable part of monitoring process

8.2 Phase 2 - Network Monitoring (Monitoring Framework)

The monitoring framework is very similar to original software architecture of the FlowMon probe. New features caused by flexible flow record are available.

The monitoring framework contains software, documentation and scripts and must be installed on the probe. The user can customize the flow record structure, generate the firmware on dedicated server, download it and install it on the probe. Now it is possible to start initialization process and to run monitoring process itself.

There are two ways how to use and control the probe - either terminal command line interface or remotely via web front-end. If the command line interface is chosen then user connects to the probe over SSH and does all operations with the probe through scripts and other programs executed directly from command line.

On the other hand, web front-end is more friendly and helps inexperienced user to configure the probe step by step. The system then consists of:

- web front-end on a remote computer with running web server
- configuration daemon on the probe
- NETCONF system for communication

It is necessary to install the web front-end on a computer with Apache web server, select NETCONF configuration on the probe and then it is possible to control the probe through the front-end. User can control several probes through one front-end.

First of all a user connects through the web front-end to the probe and is informed about current firmware version and probe configuration. He can select another of installed firmware for using or choose downloaded firmware package for installation. The installation from the package is done automatically and the new firmware version is offered to use since this time.

The firmware is booted into the COMBO cards after its version selection for using. It is initialized and characteristic of the probe (timeouts, samplings etc.)

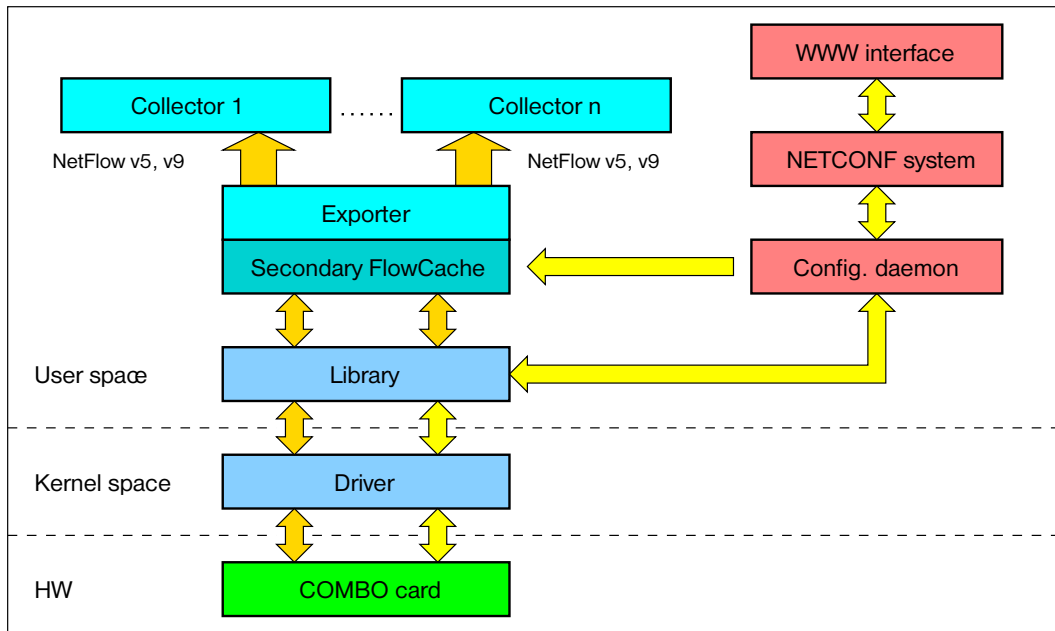


Figure 6: Software layers - remote configuration

are set to default values (startup configuration). User can reconfigure any of the characteristic of the probe including specification of the export protocol and collectors for sending NetFlow records.

User is working with two types of configurations - startup and running - during the probe configuration. Any changes in running configuration are immediately reflected on the probe in comparison with startup configuration that is realized just after restart of the probe. The advantage of this principle is in possibility to get back to functional configuration when problem occurs with a new configuration.

8.3 Secondary Flow Cache in Software

A great change in monitoring process against current version of the probe is in usage of a secondary flow cache in software. Size of a flow cache is one of the most important parameter of the probe during monitoring high speed networks. Because it is not possible to still extend the cache in the firmware so it is supported in software. Expired flow records come from flow cache in firmware to secondary flow cache in software. The expiration is done because of timeouts, collisions or flow cache capacity.

In secondary flow cache is decided about next aggregation of the flows with corresponding flows or about providing them to exporting program for export to collectors. Secondary flow cache in software works with the flow records for

aggregation in the same way as the flow cache in firmware and its expiration is based on the same timeouts. The secondary flow cache in software rapidly increases limits of the probe in monitoring high numbers of flows on high speed networks.

9 Performance Analysis

The concept of the probe is based on the processing of the incoming traffic at the speed of ten gigabit in the firmware where it is decelerated so the outgoing data stream can be handled in software. From our experience, the host PC with common network interface card is able to monitor up to 800'000 packets per second. If a specialized card with optimized DMA/PCI transfers to PC memory is utilized then it is able to process up to 1.5 million packets per second which equals to one gigabit per second (64 bytes packets). For our purpose it means that no matter of what is the incoming data rate it must be reduced by aggregation to value lower than one gigabit. In current firmware architecture the aggregation is limited by available capacity for flow records. Moreover, the aggregation factor is influenced by direct addressing of flow records with hash value. In fact the memory capacity would be probably sufficient to hold all simultaneous flow records if there was a perfect hash that could distinguish all flows producing result of bit-width the same as an address.

Of course such hash function cannot exist. The probability of collision for direct addressing can be approximately expressed as

$$P_{\text{collision}} = N/C$$

where N is current number of flow-records in memory and C is the capacity of memory.

Today a typical ten gigabit traffic consists of hundreds of thousands flows (it may vary depending on the network). Therefore the probability of collisions is very high for small memories. Despite that even small memory can provide certain level of aggregation. It is due to the burstiness charter of the traffic and the fact that 10%-20% of flows account for 90% of total traffic.

The behavior of the firmware was simulated on model written in Perl. Several samples of real traffic on ten gigabit link collected by Sven Ubik were simulated and deceleration factor was derived as ratio of total number of packets to expired flow records (size of the flow record is 64 bytes). Figure 7 shows deceleration factor as a function of memory size. The course of function is linear.

Graph in Figure 8 shows how many flows are created when using different sizes of memories. Again smaller memories create a lot of flows (so called fragmented-

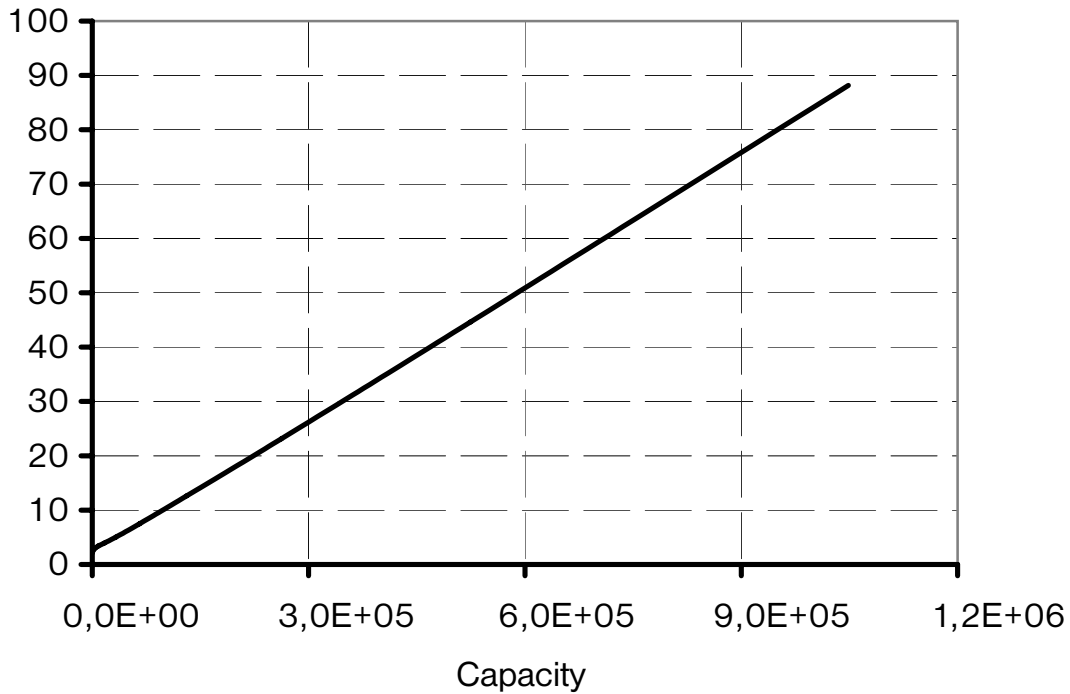


Figure 7: Deceleration factor in dependence on the size of memory

flows) because of high number of collisions. As the size of memory increases, the number of collisions decrease and the number of total flows is stabilized. The situation is shown for two settings of inactive timeout which influences number of flow records in the memory and thus the probability of collision. Higher timeout also put together those flows that are by shorter timeout marked as inactive. When the number of created flows remains steady it shows us that the number of real existing flows in the traffic is reached and majority of flows is not fragmented.

So far we were interested in the behavior of the probe as if it has unlimited throughput. It allows to model the worst case scenario from host PC point of view. For example, if the throughput on the card is not sufficient to process ten gigabit traffic then the outgoing stream to the PC is reduced consequently and therefore lower than expected by simulations. The throughput of proposed firmware architecture is limited by throughput of memory for flow records which differs according to type of the memory. There are internal BlockRAM memories in FPGA or external QDR SSRAM and SDRAM memory on the card. The BlockRAM memories are very fast but its joint capacity is insufficient to decelerate incoming traffic. They can accommodate about eight thousand flows, thus the reduction ratio is too small (about 3 times according to simulations). Two QDR SSRAM are high throughput external memories with joint capacity for 256K of flow records (deceleration factor more than 20 times). QDR SSRAM

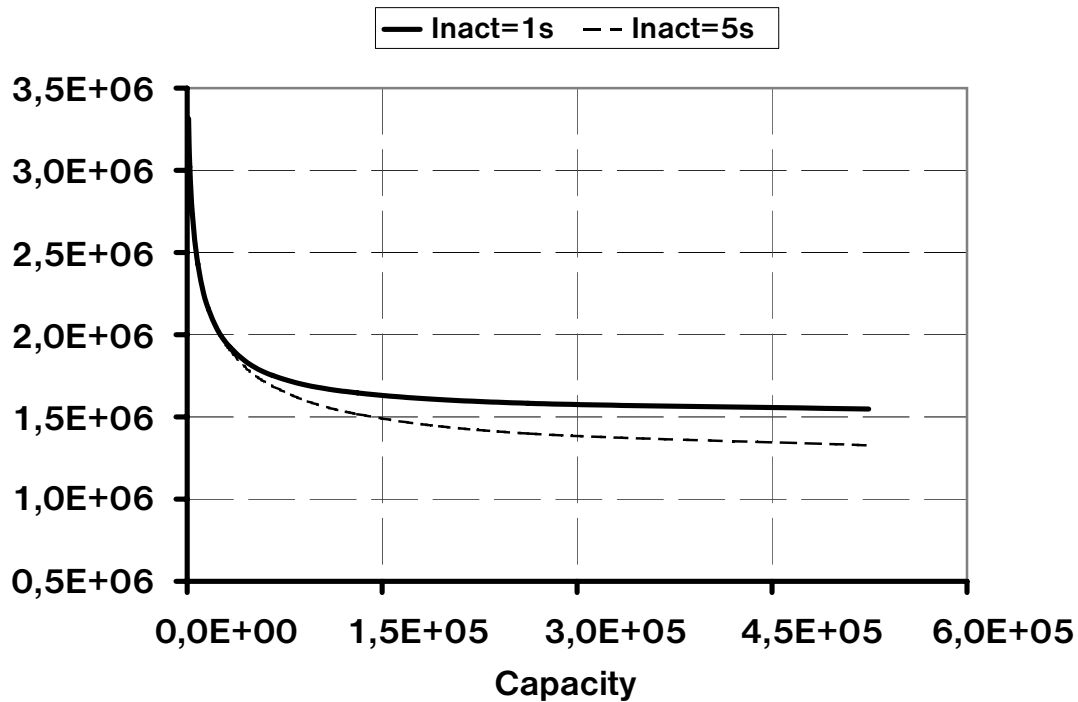


Figure 8: Number of seen flows in dependence on the size of memory

are equipped with extra write and read interfaces, each interface has ten gigabit throughput which correlates with ten gigabit data rate of incoming traffic. Time information for Flow State Manager can be stored in Block Rams. The last option is to use DRAM memory as the memory for the flow records which gives a high deceleration factor (about 250) but the throughput of the memory is not sufficient (about six gigabits and only one shared port for reading and writing).

The analysis shows that if the firmware can reach the optimal memory bandwidth utilization (might be challenge) the whole probe will be able to process full ten gigabit link without a loss of packet.

10 Conclusion

Network monitoring based on flows is very popular. Collected statistics describes meta data of the network unlike payload monitoring which is focused to collect whole packet traces. Mostly used NetFlow export format is NetFlow v5, introduced by Cisco. While it served well, growing requirements on flow monitoring caused new protocols to emerge, e.g., NetFlow v9 or IPFIX. These protocols support modification of the flow record in the way that only required information (from the list [Cla07]) is exported.

The report presented a framework to define customized flow record with assigned monitoring operation to each item of the flow record. It allows the user to customize its monitoring process and to acquire nearly any information out of the network traffic. There are several possible applications suggested, based on our experience from flow monitoring and also based on survey through out several research papers.

It seems that there is a lack of dedicated hardware probes that can report relevant data for suggested application. Therefore we propose flexible flow monitoring probe (fFlowMon) based on commodity PC and hardware acceleration card. The architecture of the probe is described together with the procedure how the flexible configuration can be reached. At the end, the performance of the suggested architecture is analyzed. The results shows that the probe should be able to process ten gigabit traffic without a packet loss.

Our future work is focused to finish the fFlowMon probe implementation and to extend its capabilities by adding extra features like application decoder. After that the probe should be deployed on real networks where we would like to test its capabilities and performance. A challenging task would be to find or implement collector that can understand all data exported by our probes.

References

- [BP01] Barford P., Plonka D. Characteristics of network traffic flow anomalies. In: ACM IMW (Nov. 2001).
- [Cla04] Claise, B. (Ed.) *Cisco Systems NetFlow Services Export Version 9*. RFC 3954³, IETF, October 2004.
- [Cla07] Claise, B. *IPFIX Protocol Specification*. Internet-Draft, work in progress, draft-ietf-ipfix-protocol-26.txt, 2007.
- [CS04] Clark Ch., Schimmel D.: Scalable Pattern Matching for High-Speed Networks. In: *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, p. 249-257, Napa, California, 2004.
- [Cro07] Crotti M. et al.: Traffic classification through simple statistical fingerprinting. *SIGCOMM Comput. Commun. Rev.* 37(1):5-16, 2007.
- [DMM07] Dedek T., Marek T., Martinek T. High Level Abstraction Language as an Alternative to Embedded Processors for Internet Packet Processing in FPGA. In: *2007 International Conference on Field Programmable Logic and Applications*, Amsterdam, IEEE CS, 2007, p. 648-651.

³<http://www.ietf.org/rfc/rfc3954.txt>

- [DWF03] Dewes C., Wichmann A., Feldmann A. An analysis of Internet chat systems. In: *IMC 03: Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement*, p. 51-64, Miami Beach, FL, USA, October 2003.
- [Dh04] Dharmapurikar S. et al.: Deep Packet Inspection using Parallel Bloom Filters. *IEEE Micro* 24(1):52-61, 2004.
- [Dre06] Dreger H. et al.: Dynamic application-layer protocol analysis for network intrusion detection. In: *Proceedings of the 15th Conference on USENIX Security Symposium*, Volume 15, p. 18. USENIX Association, Berkeley, CA, 2006.
- [GMT05] Gu Y., McCallum A., Towsley D.: *Detecting anomalies in network traffic using maximum entropy*. Tech. rep., Department of Computer Science, UMASS, Amherst, 2005. Available online⁴.
- [KSV07] Kompella R. R., Singh S., and Varghese G. 2007. On scalable attack detection in the network. *IEEE/ACM Trans. Netw.* 15(1):14-25.
- [KK07a] Kořenek J., Kobierský P. Intrusion Detection System Intended for Multigigabit Networks. In: *DDECS 2007*, p. 361-364
- [KK07] Košek M., Kořenek J. FLOWCONTEXT: Flexible Platform for Multigigabit Stateful Packet Processing. In: *2007 International Conference on Field Programmable Logic and Applications*, Amsterdam, IEEE CS, 2007, p. 804-807.
- [MT06] Martinek T., Tobola J. *Interconnection System for the NetCOPE Platform*. Technical Report 34/2006⁵, CESNET, 2006
- [McG04] McGregor A. et al.: Flow Clustering Using Machine Learning Techniques. In: *Proceedings of the 5th Passive and Active Measurement Workshop (PAM 2004)*, p. 205-214, Antibes Juan-les-Pins, France, March 2004.
- [Mol04] Molina M. et al. Design principles and algorithms for effective high speed ip flow monitoring. *Computer Communications* 29: 1653-1664, 2006.
- [Moo03] Moore, A. et al. Architecture of a Network Monitor. In: *Passive & Active Measurement Workshop 2003*. Available online⁶.

⁴<http://citeseer.ist.psu.edu/gu05detecting.html>

⁵<http://www.cesnet.cz/doc/techzpravy/2006/netcope-interconnection/>

⁶<http://citeseer.ist.psu.edu/moore03architecture.html>

- [MP05] Moore A. W., Papagiannaki K.: Toward the Accurate Identification of Network Applications. In: *Proceedings of the 6th Passive and Active Measurement Workshop (PAM 2005)*, p. 41-54, October 2005.
- [MZ05] Moore A. W., Zuev D.: Internet traffic classification using bayesian analysis techniques. In: *SIGMETRICS 05: Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, p. 50-60, Banff, Alberta, Canada, June 2005.
- [Pa94] Paxson V. Empirically derived analytic models of wide-area TCP connections. *IEEE/ACM Trans. Netw.*, 2(4):316-336, 1994.
- [Qui04] Quittek, J. et al. *Requirements for IP flow information export (IPFIX)*. RFC 3917⁷, IETF, October 2004.
- [TBS06] Tan L., Brotherton B., Sherwood T.: Bit-split string-matching engines for intrusion detection and prevention. *ACM Trans. Archit. Code Optim.* 3(1):3-34, 2006.
- [Tru05] Trussell H. J. et al.: Characterization, Estimation and Detection of Network Application Traffic. In: Proc. *EUSIPCO 2005*, Antalya, Turkey, 2005.

⁷<http://www.ietf.org/rfc/rfc3917.txt>