

# **Hardware-Accelerated NetFlow Probe**

**Martin Žádník and Ladislav Lhotka**

14. 12. 2005

## **1 Abstract**

With ever-growing volume of data being transferred over the Internet, the need for reliable monitoring becomes more urgent. Monitoring devices should be able to provide accurate information such as traffic patterns, statistics and various anomalies. This technical report describes an implementation of a network flow monitoring system using a dedicated hardware platform cooperating with the host PC. By exploiting the hardware-software codesign principle, we implemented time-critical functions in hardware and the rest in software. This way, the NetFlow probe offers good performance at low cost. After explaining the basics of flow measurement techniques we describe the design of the probe and function of the individual units. Finally, results and future work are also discussed.

## **2 Introduction**

Most modern communication services (world wide web, streaming, databases, e-mail, on-line shops etc.) now use the Internet infrastructure. Its reliable operation also depends on large-scale monitoring capable of providing accurate data about traffic patterns, applications used, hostile activities etc. Such monitoring systems can help network operators to manage their current networks or plan new network topologies. Other management techniques such as bandwidth provisioning, detecting DoS attacks, billing and accounting also require detailed monitoring. Currently available monitoring devices have their limitations in terms of performance and flexibility. In particular, for security-related applications it is not acceptable to get information about only a random portion of the network traffic when the monitoring device becomes overloaded, for example during a DoS attack.

NetFlow as a general method for flow monitoring [RFC3954], first implemented in Cisco routers, is the most widely used measurement solution today. Statistics on IP traffic flows provide information about who communicates with whom,

how long, how often, using what protocol and service and also how much data was transferred.

So far, Netflow data are usually acquired and exported by IP routers. Such a setup has several drawbacks, namely

- Routers are by definition visible Layer 3 systems that can easily be discovered by simple tools such as *traceroute*. Consequently, they can become targets for all types of attacks.
- The main task of routers is to forward datagrams and exchange routing information with their neighbours. The processing power available is thus rather limited and so operations on Netflow data do not usually go much beyond simple export of raw flow records.
- As a special case of the previous item, some routers impose sampling on the incoming traffic. Even if sampling is not strictly required, in some cases it is the only way for keeping the router operational, especially when high-speed interfaces are monitored.

In contrast, our standalone monitoring probe is essentially a stealth device – invisible at both Layer 3 and 2 – dedicating all its resources to the tasks of flow record acquisition and processing.

## **2.1 Flow**

A IP traffic flow is a set of packets with common properties that passes an observation point. Those common properties can be any information contained in packet headers or related to the packet [RFC3917].

## **2.2 Flow characteristics**

From the definition of IP flow it is clear that we can always obtain different flows by applying different views on the same set of packets. For example, aggregation according to addresses, ports and sequence numbers results in one packet per flow. On the other hand, when using IP version as the only distinguishing property, we get just two flows, one for IPv4 and another for IPv6. The selected properties thus influence the size and duration of flows.

For some transport protocols, the end of a flow can be determined from specific events such as the presence of the FIN flag in the TCP header [RFC3917]. For protocols that do not provide such an indication, two timeouts are used for terminating the flows. The *inactive timeout* controls how long the flow is kept in memory when no incoming packets belonging to the flow are seen. On the other hand, the *active timeout* defines the time period with which long-lasting flows are artificially terminated and information about them exported.

## 2.3 Sampling

Nowadays, the Internet traffic has no lack of malicious traffic such as DoS attacks, smurfs and port scans. Such traffic often generates a large number of flows. Consequently, NetFlow monitoring systems may become overwhelmed and thus unable to give vital information about those attacks. Many techniques were published aimed at protecting the monitoring elements so that they continue operation even when faced with massive volumes of malicious traffic. Examples are:

**Input sampling:** Input sampling of incoming packets is the easiest way how to decrease the traffic volume to be processed, and also decrease the number of new flows during attacks where every incoming packet belongs to a new flow. On the other hand, sampling makes it difficult to estimate precise flow statistics.

**Output Sampling:** Rate of exported flow records is strongly dependent on the actual traffic mix. Output sampling can keep it in a specified range and prevent collector from being overwhelmed by aggressive export rate during traffic peaks or attacks.

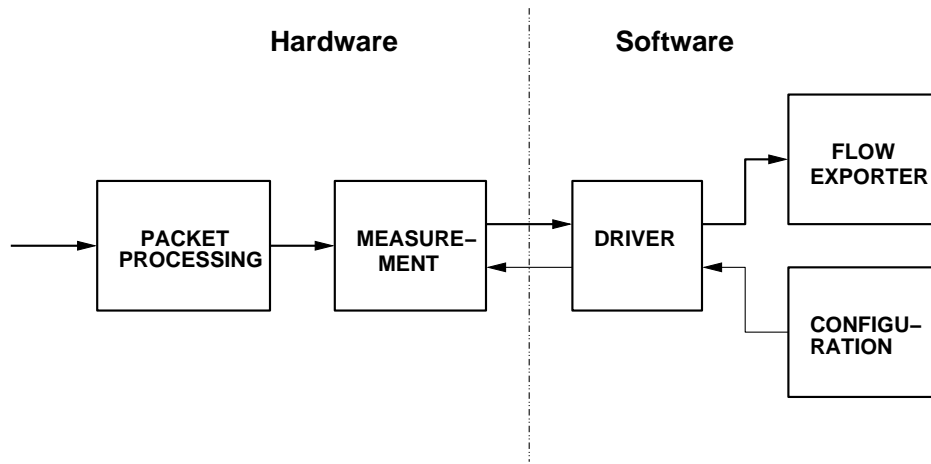
**Sample and Hold:** This method is quite similar to input sampling but with the following twist. As with ordinary sampling, each packet is sampled with certain probability. However, for every new entry in the flow memory, all subsequent packets belonging to that flow are protected from sampling the flow memory, a new item is created. This way we obtain precise data about large flows [Es03].

**Adaptive Input Sampling:** A static sampling rate is either suboptimal at low traffic volumes or can exhaust resources (memory, bandwidth) or cause other difficulties at high traffic volumes. Adaptive Input Sampling keeps the device load within reasonable limits while using the optimal sampling rate for all traffic mixes.

**Matching:** Only those packets that meet certain rules are sampled.

## 3 Block structure of the Netflow probe

The Netflow probe consists of two parts: hardware and software. The hardware part processes incoming packets at high speed while the software together with an ordinary network card handle the task of transmitting flow records to a remote collector. This division of functions (Figure 1.1) guarantees very good performance and flexibility in processing methods and export formats.

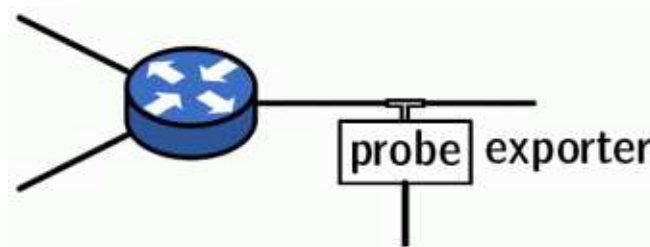


**Figure 1:** Partitioning of system

## 4 Hardware/firmware characteristics

As a hardware platform platform we use the COMBO cards developed by the Liberouter project [CoHW]. COMBO6 is a universal PCI card, which can be used in various applications. It consists of Xilinx Virtex-II XC2V3000 FPGA, 2MB Ternary CAM, 256MB DRAM and three SSRAM chips, each with 2MB of memory. Various add-on cards can be used with the COMBO6 card, for example the COMBO-MTX or COMBO-SFP interface cards. Additional cards are now being developed with more on-board memory and a variety of network interfaces (GE, 10GE, PoS STM-16). With necessary modifications, the flow data acquisition firmware will be able to use the new interface cards as well.

During one year we have developed a fully functional prototype. The firmware is somewhat limited by the interface types and resources available on the current COMBO cards: only Gigabit Ethernet interfaces are supported and the maximum number of flows that can be monitored simultaneously is 65536.

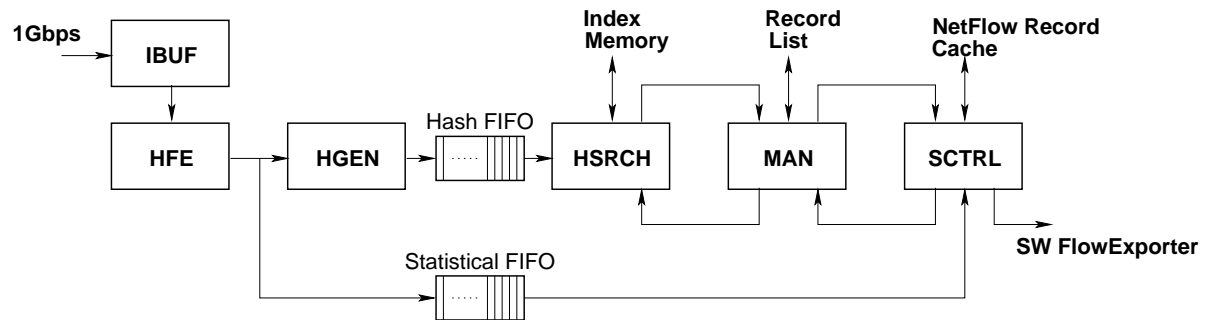


**Figure 2:** NetFlow probe as repeater

The probe works as a T-splitter (see Figure 2 2 ): when inserted into a network link, the traffic is passed directly to the original destination and a separate copy

of link data is processed by the probe in parallel. From the network perspective, the probe can be classified as a repeater that is invisible at both the network and link layer.

The copy of link data is processed using an FPGA (Field Programmable Gate Array). FPGAs allow us to create our own application-specific architecture. The block diagram of the NetFlow firmware architecture is in Figure 3.3. Characteristics important from the network operator viewpoint are discussed in the following paragraphs. We describe only those units that are crucial for understanding the operation of the probe.



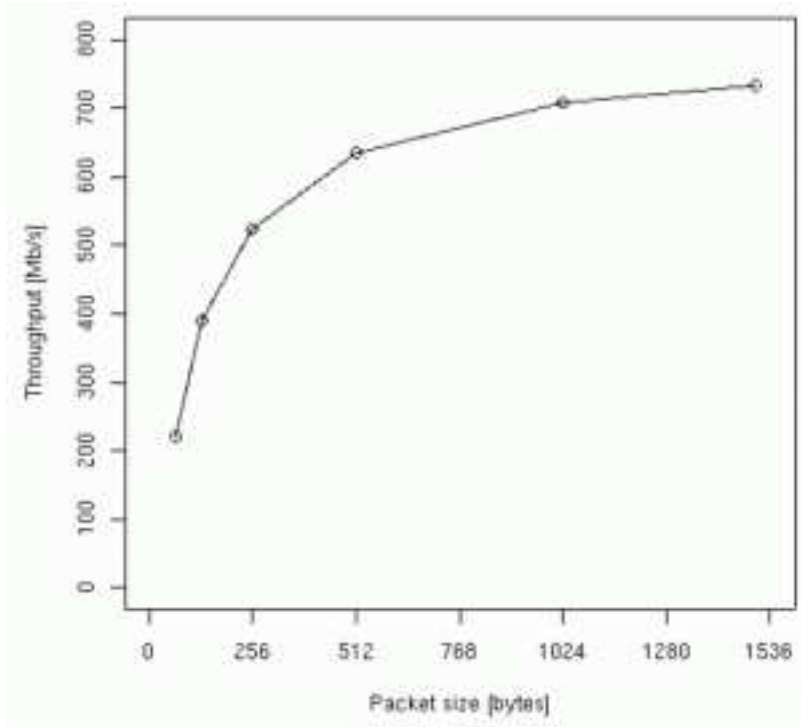
**Figure 3:** Block structure of firmware architecture

The firmware is able to process IPv4 and IPv6 datagrams of all sizes in full 1 Gbps line rate. CRC checksum is verified and a 32-bit timestamp with the precision of 640ns assigned. In the Input Buffer (IBUF), numbers of bad and dropped packets as well as overall statistics of the incoming traffic packets are recorded. We plan to implement both static and adaptive input sampling in the near future.

In the Header Field Extractor unit (HFE), headers of incoming packets are parsed and a configurable set of "interesting" header fields, flags and other characteristics are extracted. These are typically source and destination IP addresses, source and destination ports, protocol number, actual packet length, TCP flags, DSCP code point, ICMP options and so on. At present, this unit is the bottleneck of the processing pipeline (see Figure 4.4) and we are thus working on a new HFE implementation with considerably improved performance.

We use hash values as unique identifiers of flow records. By default, five header fields are used as inputs for the hash function CRC-64 implemented by the HGEN unit: two IP addresses, two IP ports and the protocol number. If necessary, selected bytes of these five key fields can be masked out in order to get more aggregated flows. We use the first 57 bits from the 64-bit hash value as the flow identifier.

For indexing and scanning the flow cache we use an index vector of  $2^M$  pointers to lists that are able to store up to eight flow records each (see Figure 6.6). We use



**Figure 4:** HFE throughput

M bits of previously computed hash value for finding the appropriate list. The rest of the hash value is stored in the flow record for fast comparison against new flows. The CRC-64 hash function distributes the flow records uniformly among the available  $2^M$  positions. Applying the standard M/M/N queuing theory, we can express the probability that the some list becomes full as [Mo05]:

$$P(\text{list} > N) = \left( \frac{1}{\sum_{k=0}^N \frac{g^{-k}}{k!}} \right) \cdot \frac{g^{-N}}{N!}$$

**Figure 5:** Probability of list overflow

where N is the length of the list, X maximum capacity of the flow cache and  $g = 2^M / X$ .

In our case we have  $N = 8$  and  $g = 2^{15} / 2^{16}$ . So  $P(\text{list\_full}) = 8.5948 \times 10^{-4}$ .

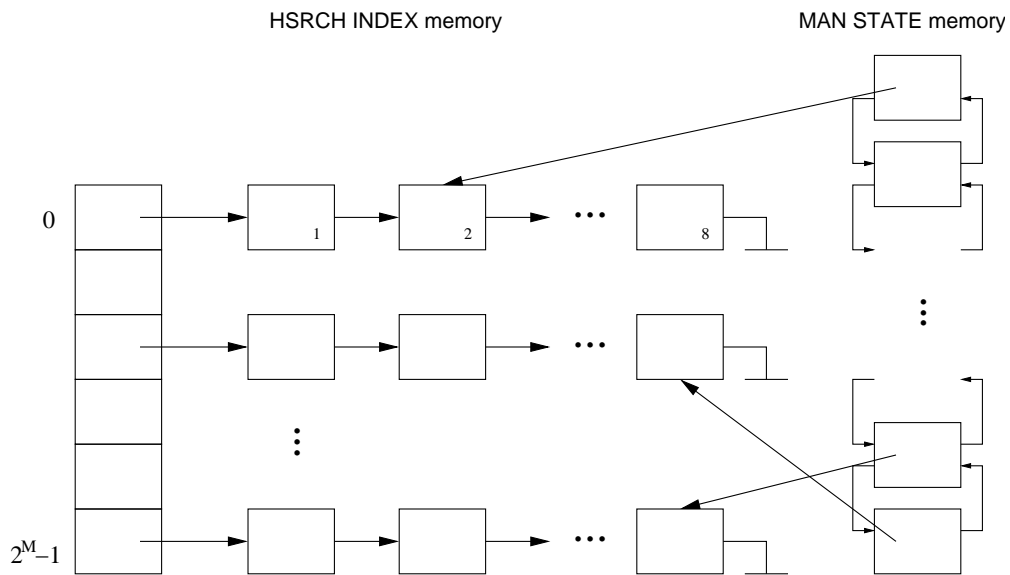
It is up to the operator to decide what to do when the list becomes full. He can instruct the device to drop the incoming packet that would not fit to the list or to export some record(s) from the list and replace it with a new record about

the incoming packet. We plan to increase the size of the flow memory and thus decrease the probability of list overflow.

The use of hashing implies that different flows may occasionally map to the same key. However, the probability of such an undetectable collision is very small – its upper bound is

$$P(\text{collision}) = X / 2^L,$$

where  $X$  is maximal number of flows in the memory and  $L$  the length of the hash value. This gives for  $X=2^{16}$  and  $L=57$  the value  $P(\text{collision}) = 4.54 \times 10^{-13}$ . This means a collision appears less than once a week on the average. The hash function is randomly initiated every time the device is started so that the hash values cannot be predicted. We also plan to add provisions for detecting flow collisions based on checking the original key field.



**Figure 6:** Memory format

The states of flow records are kept in a bidirectional list. The list is sorted by timestamps. We can set very precise timeouts: the inactive timeout in the range 0-60s with a precision of 10 microseconds and the active timeout in the range 0-1200s with a precision of 1 microsecond.

Traffic sampling is always optional. Apart from the standard static input sampling we also implemented the sample-and-hold technique. Two sampling methods are implemented:

- regular sampling (the default method) means that every  $N$ -th packet is taken, and



1	length 0 - OK, 1 - longer than MTU
2	RUNT 0 - OK, 1 - shorter than 64 bytes
3	IP 0 - IP, 1 - NonIP
4	IPver 0 - IPv4, 1 - IPv6
5	SRC_PORT validity 1 - SRC_PORT valid, 0 - SRC_PORT doesn't contain valid data
6	DST_PORT validity 1 - DST_PORT valid, 0 - DST_PORT doesn't contain valid data
7	BAD_ETH 0 - frame seems ok, 1 - ethernet frame is bad (VLAN, ...)
8	OTHER_ERR: 0 - good, 1 - other unspecified error
9	802.1Q tag present
10	MPLS unicast
11	MPLS multicast
12	TCP flags are valid
13-15	reserved

*Hash Generator* (HGEN) takes the key fields and feeds them to the CRC-64 function. The resulting hash value serves as the flow identifier. HGEN allows to process 32 bits of input in one tick at 100MHz, so the throughput is 3.2Gbps.

The *Hash Search* (HSRCH) unit searches the flow cache for the entry that matches the generated hash value. This operation can end up with several different results:

- No matching entry is found so a new one is created.
- No matching entry is found but the flow cache is full.
- A matching entry is found.

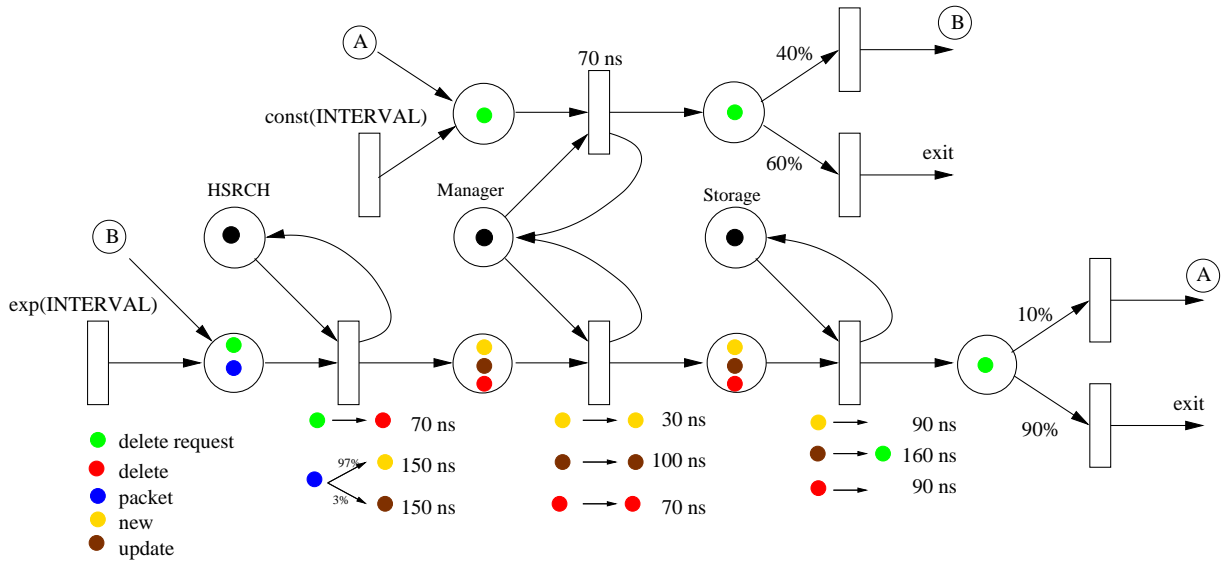
The search result and additional data are transferred to the *Manager*. It is also necessary to remove expired entries according to requests from *Manager* and acknowledge it back.

The *Manager* is responsible for keeping states of active and inactive flows. For this purpose bidirectional linked list is used. New or updated flows are inserted at the top of the list so that the oldest ones gradually sink to the bottom. It is then easy to identify inactive flows and send delete request to HSRCH. Another task of the *Manager* unit is to copy information coming from *Hash Search* to the *Storage* unit and vice versa. Since *Storage* can also generate delete requests, *Manager* must be careful to avoid duplicate requests to *Hash Search*.

Statistical data is held in *Storage* memory. The unit reads data from *Statistical FIFO* and performs different operations according to commands obtained from *Manager*. The unit is responsible for checking whether records are not held for too long (starting timestamp of the flow is compared to the actual timestamp while flow statistics are updated). If it is the case, a delete request is sent to *Manager*.



throughput of the design with different traffic mixes. The length of each queue was monitored to find out whether the corresponding unit is able to process all requests. A simplified view of the model based on Petri Net is shown in Figure 7 7.



**Figure 7:** Petri Net model of the flow processing firmware.

Our model was implemented in C++ using the SIMLIB simulation library.

*Experimental setup:*

With memory tables filled to 50 percent of their capacity we tried to simulate an attack by flooding the probe with a large number single-packet flows during one second.

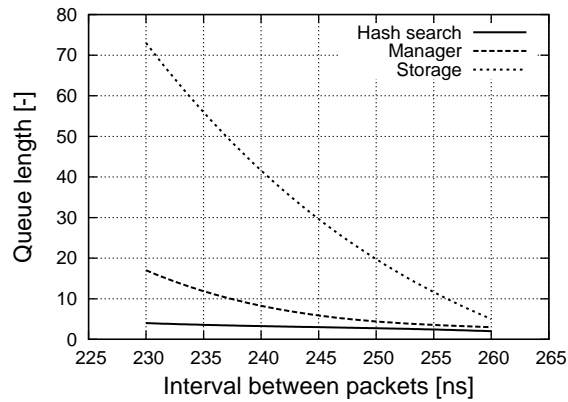
*Results:*

After several simulation cycles for different inter-packet intervals we have obtained the graph in Figure 8 8. Simulation results show that our probe should be capable to process 3 to 4 million packets per second, which is equivalent to 260ns interval between packets.

## 6.2 Collisions

We randomly generated *Unified Headers* and processed them with the same CRC-64 function as implemented in the firmware. The *Hash Search* memory lookup procedure was simulated to find out the maximum length of index lists. The results are in Figure 10 10. Also the number of collisions caused by CRC reduction was counted.

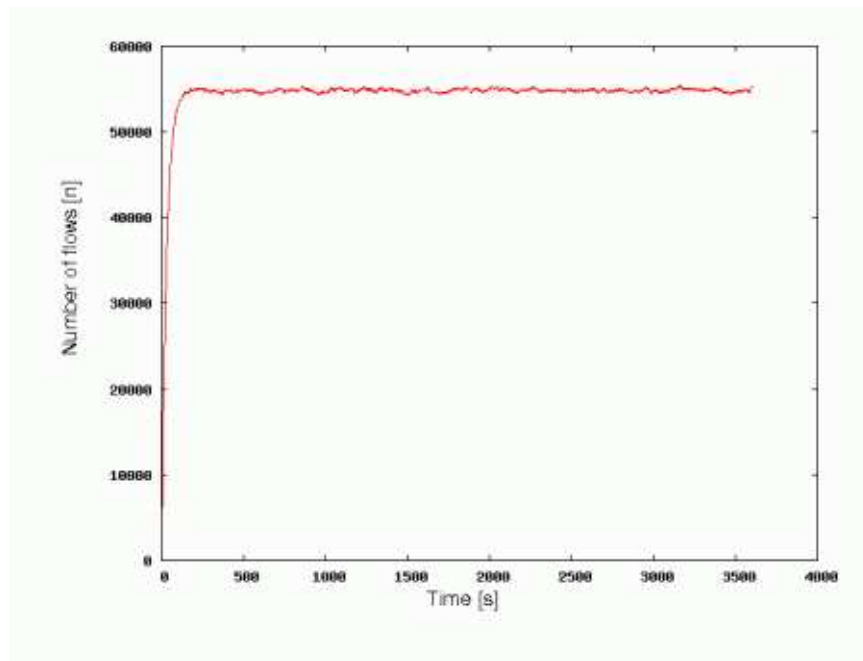
*Experimental setup:*



**Figure 8:** Maximal length of queues during simulation

We generated randomly packet headers with uniform distribution. Packets arrivals were simulated as a Poisson process and exponentially distributed lifetime of flows in the memory was used.

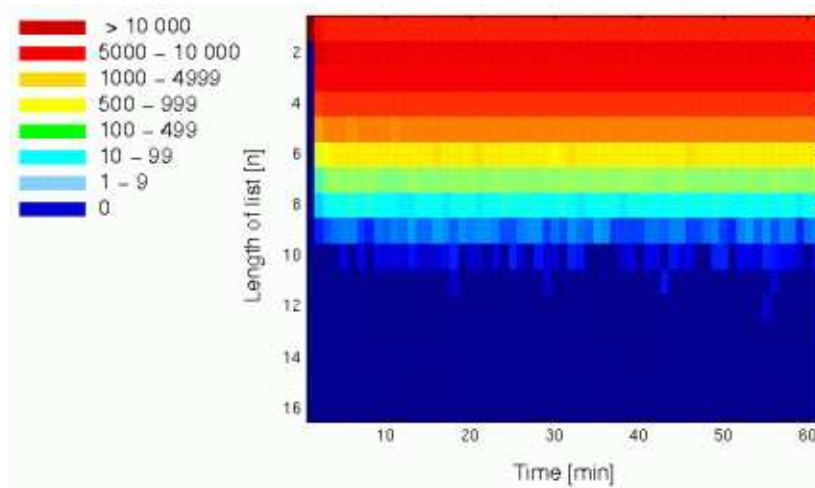
The model started with empty flow memory and each simulation consisted of more than fifty thousand flows. Number of flows in the memory during the simulation is shown in Figure 9 9. Model time was sixty minutes and it took 15 hours with Pentium 4 running at 2 GHz to finish the simulation.



**Figure 9:** Number of flows during simulation

## Results:

First, we were interested in the number of collisions caused by the method chosen for indexing the  $2^{15}$  lists of 8 flow entries. Collision in this case means that any of the lists becomes full (contains more than eight items). Figure 10 shows the distribution of lists lengths in the course of the simulation.



**Figure 10:** Number of lists dependent on the occupation and time

Next, during this experiment we did not observe any single collision due to CRC space reduction.

## 7 Conclusions

A functional prototype of the probe was tested on the access link connecting the Brno Academic Computer Network to the CESNET2 backbone. The results so far are very promising.

The probe firmware now works at 50MHz, which results in the theoretical maximum throughput of 800 Mbit/s. With the new COMBO6X card we will be able to double the clock rate and together with the improved implementation of the *Header Field Extractor* this should be more than enough for wire speed processing of Gigabit Ethernet links. The flow cache currently has room for 64 thousand flows but this capacity will soon be increased to 512 thousand flows.

The software part of the probe is described in another technical report [Ce05].

We will port the flow processing firmware to the new more powerful cards such as the COMBO6E/COMBO6X motherboards and COMBO-2XFPRO and COMBO-4SFPRO interface cards. That will allow us to monitor 10GE and PoS STM-16 links,

although input sampling will be necessary for traffic rates exceeding 1.6 Gb/s. The almost ten-fold increase in flow cache capacity will also be an important improvement – the flow cache should then be able to absorb even the hardest known DoS attacks. We also plan to enhance the design in the direction of IPFIX compliance by means of supporting variable record format, additional sampling techniques and so on . . .

## References

- [Ce05] Čeleda, P., Kováčik, M., Krejčí, R., Kysela, J. and Špringl, P. *Software for NetFlow Monitoring Adapter*. Technical Report XX/2005, CESNET, Praha, 2005.
- [Es03] Estan, C., Varghese, G.: New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. *ACM Trans. Comput. Syst.* 21, 2003.
- [RFC3917] Quittek, J., Zseby, T., Claise, B. and Zander, S. *Requirements for IP Flow Information Export (IPFIX)*. RFC 3917, IETF, 2004.
- [RFC3954] Claise, B. (Ed.). *Cisco Systems NetFlow Services Export Version 9*.
- [Mo05] Molina, M., Chiosi, A., D’Antonio, S. and Ventre, G.: Design principles and algorithms for effective high-speed IP flow monitoring. *Computer Communications*, in press.
- [CoHW] Liberouter Project. Description of COMBO cards. <http://www.liberouter.org/hardware.php>
- [Ha05] Unified Header Specification for Net-Flow, [http://www.liberouter.org/cgi-bin2/cvsweb.cgi/liberouter/vhdl\\_design/units/hfe/doc/UH-Netflow.txt?rev=1.18](http://www.liberouter.org/cgi-bin2/cvsweb.cgi/liberouter/vhdl_design/units/hfe/doc/UH-Netflow.txt?rev=1.18)