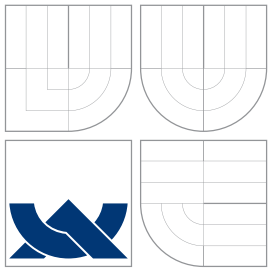
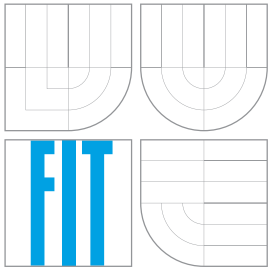


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

NÁVRH ARCHITEKTURY SONDY PRO MONITOROVÁNÍ SÍŤOVÝCH TOKŮ

DESIGN OF FLOW MONITORING PROBE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. MARTIN ŽÁDNÍK
AUTHOR

VEDOUCÍ PRÁCE — Ing. JAN KOŘENEK
SUPERVISOR

BRNO 2007

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. Pan

Jméno a příjmení: **Bc. Martin Žádník**
Id studenta: 49297
Bytem: Staměřice 94, 751 25 Dolní Újezd
Narozen: 25. 10. 1982, Přerov
(dále jen "autor")

a

2. Vysoké učení technické v Brně

Fakulta informačních technologií
se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....
(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
diplomová práce

Název VŠKP: Návrh architektury sondy pro monitorování síťových toků
Vedoucí/školitel VŠKP: Kořenek Jan, Ing.
Ústav: Ústav počítačových systémů
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:
tištěné formě počet exemplářů: 1
elektronické formě počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti:
   - ☐ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☒ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

.....	.....
Nabyvatel	Autor

Abstrakt

Tato práce popisuje návrh a implementaci monitorovací sondy pro sběr statistických údajů o tocích na vysokorychlostních sítích. Navržená sonda využívá spojení hardwarové akcelerační karty a běžného PC. Úloha zpracování a monitorování paketů na rychlosti deset gigabitů je distribuována mezi kartu a hostitelský počítač. Z hlediska systémové architektury jsou zkoumány algoritmy pro monitorování toků a navrženy způsoby jejich implementace v hardwaru či softwaru. Dále jsou navržena rozšíření sondy o možnosti autokonfigurace a adaptace řídích parametrů na základě aktuálního vytížení sondy. Významným rozšířením je konfigurovatelná definice sledovaných údajů o toku. Výsledky simulací potvrzují vysoký výkon a robustnost sondy pro monitorování desetigigabitových sítí.

Klíčová slova

Síť, monitorování, IP tok, NetFlow, architektura, model, simulace.

Abstract

This thesis deals with the design and implementation of a monitoring probe intended for IP flow measurements in high-speed networks. The probe is based on commodity PC and network acceleration card. The monitoring process is partitioned between these two platforms. The thesis explores ways of mapping flow monitoring algorithms to hardware or software implementations. Several improvements are suggested to increase performance and functionality of the probe. Two level memory hierarchy increases the performance whereas autoconfiguration and adaptation of control parameters contribute to its robustness. The definition of variable flow-record allows to customize monitored statistics about the network. Analysis and simulations of proposed architecture indicate that the probe is suitable for monitoring of ten gigabit networks.

Keywords

Network, monitoring, IP flow, NetFlow, architecture, model, simulation.

Citace

Martin Žádník: Design of Flow Monitoring Probe, diplomová práce, Brno, FIT VUT v Brně, 2007

Design of Flow Monitoring Probe

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Jana Kořenka.

.....
Martin Žádník
19th May 2007

Poděkování

I would like to thank my supervisor Ing. Jan Kořenek for his advice and information. I am also grateful to my colleges from Liberouter project whom I discussed parts of my work with.

Contents

Chapter 1

Introduction

Information technology and closely related network technology are very fast developing fields. More data is stored and transfered in digital form which allows its fast retrieval and processing. This trend correlates with the growth of the Internet as an interconnection network of local networks. Growing number of people is connected to the Internet at work but also at home. Not only number of users grows but their demands on network capabilities are higher than was ten years ago. Nowadays user requires low latency high bandwidth interconnections to be able to transfer large amount of data, to access electronic services and to react in real-time on the network. Especially last mentioned demand is in the focus during last years and its achievement allows users and machines to respond very fast on recent events for example to deal on the exchange or to have telephone calls using Internet. It is said that who has information has also the power. So it is only natural that growing number of people access the network to use its services such as e-mail, web pages, distributed database, VoIP, Internet-banking, etc. Today we witness large networks with complex topologies working at high transfer speeds. Unfortunately we also witness how easily they can be misused if they are not managed and monitored properly.

The Internet is based on IP protocol and is composed of many network domains which are more or less administrated by different entities. It is obvious that such an environment is not too friendly for reliable services. Various types of attacks can cause denial of service, leakage of information or to increase interconnection latency. Therefore there is a need for monitoring devices which are able to provide accurate data about spectrum of traffic mix, attacks, applications, etc. Such type of systems can help network operators to manage current networks or plan new network topologies. In addition, bandwidth provisioning, detecting DoS attacks, billing and accounting are also possible and required today.

Nowadays situation in the field of network monitoring is not satisfiable. Although some active devices have monitoring abilities (added value to their original functionality) it is usually only a portion of the traffic about what they are able to give appropriate data [18]. E.g., during a traffic peak or an attack the device is overwhelmed and unable to monitor whole bandwidth. This significantly decreases the quality of statistical information it provides. Unfortunately some networks are without any monitoring systems at all. Mainly because of lack of available and suitable monitoring devices or just because of they are using devices of another vendor which does not support monitoring in his equipments. Such networks connected to the Internet significantly undermines global effort to brings the order in the world largest network and they are often source of attacks against the rest of the networks. The minimal solution is to place monitoring-capable device at the edge of the local network where it is connected to the Internet. The question is which type of moni-

toring to choose. Monitoring abilities differ from one device to another. A lot of methods have been introduced during last few decades. Each usually focuses on a specific problem. For example monitoring based on simple counters per interface is suitable when network operator wants to know the utilization of the device but it is absolutely useless for billing or payload checking. Advanced solutions are provided by devices where operators can specify rules for the traffic they want to analyze thus seeing only a portion of the original traffic. The complex solution with good abstraction is based on flows. Aggregate statistic data can be viewed from different angles and with different granularity. Also in this field some devices are optimized to gather only specific flows such as TCP connections.

General flow monitoring, first introduced by Cisco, is called NetFlow. Originally it was used as a cache to support routing decisions. Later on when the architecture changed, the flow cache was left in place for statistic purposes only and showed up pretty useful and popular. NetFlow is the most widely used measurement solution today. Keeping state for every flow allows to tell with whom, how long, at what intervals, with what protocol and port how much data was transfered [2]. Unfortunately NetFlow is Cisco proprietary protocol. Therefore Internet Engineering Task Force (IETF) introduced open standard in form of RFC dedicated to flow information export. It is called IPFIX and allows all network devices capable of collecting flow-oriented statistics to export it.

Most of the network elements capable of exporting flow information are routers. In parallel, software PC-based probes are also developed. In comparison with routers standalone dedicated device for such type of monitoring has several benefits: no need to change routers that do not support NetFlow, high speed of data processing, large flow cache, various enhancements to protect itself against malicious traffic (for example [15, 18]).

Unfortunately pure software implementations suffers from lack of resources to process whole bandwidth of current backbone links with speeds exceeding 1 Gbps (see results of tests in Annex D). Robust solutions for high-speed flow monitoring require some kind of wire acceleration. In recent years field programmable gate arrays became popular way to hard-wire specific architectures. In comparison to ASICs their configuration is not static and can be changed anytime new firmware is uploaded. This allows a rapid development, smaller time to market interval and space for bug fixing. Thus field programmable gate array placed right on the network card can be a good way to accelerate traffic monitoring. The data acquisition and statistic computation take place on the card and only results are sent to the host PC. Chip manufacturers indeed provide wide range of network FPGA boards with support of various network interfaces such as one or ten gig Ethernet.

The thesis is organized as follows. In the beginning, the work focuses on an introduction to IP network and IP network monitoring systems with emphasis on flow-oriented measurement. Next chapter describes standards of flow measurement and its export. Various techniques of flow measurement are discussed in the forth chapter. Chap. 5 describes the architecture of the probe. Several models are given in Chap. 6 in order to study the behavior of the probe. Achieved results and future work are discussed in conclusion.

Chapter 2

Theoretical Background

In the Internet, when two network nodes want to communicate with each other they wrap data into several packets and send them on the network interface. Packets traverse via networks to the destination points which is fairly complex process.

Therefore if we want to obtain accurate monitoring results we must understand all underlying mechanism, protocols and events that influence communication on the network. In other words we have to know what do we work with, what to measure, how to do it and how to interpret the results. Following paragraphs answer first two points by describing standard Internet traffic model TCP/IP focusing in details on important features of IP flows monitoring. After that basic principles of network monitoring are described.

2.1 TCP/IP Model

The TCP/IP is a derivation of reference model ISO/OSI for Internet. Unlike ISO/OSI it has lower expectation of the underlying network, above all it does not suppose any reliable transfer till the application layer. The only demand is best effort delivery. The model has four layers and their approximate equivalents to ISO/OSI models are denoted in Figure 2.1.

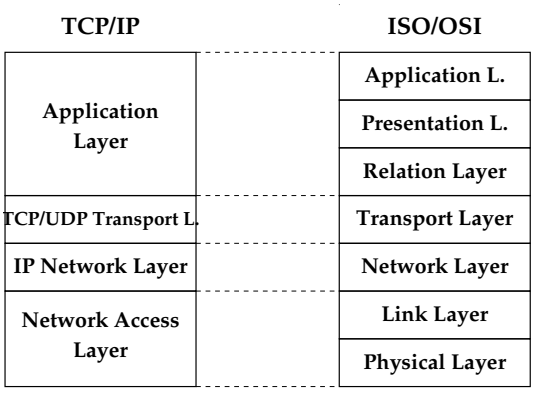


Figure 2.1: TCP/IP model used in Internet communication and ISO/OSI reference model

The purpose of these layered models is to provide various levels of abstraction and to arrange tasks into separate categories. When the packet travels across the networks its content is analyzed at different layers. Each layer has its own function and provides specific information for network nodes to transfer packets to the destination point correctly.

Following sections describe each layer in greater detail and specifies its functionality.

2.1.1 Network Access Layer

TCP/IP groups physical and link layer into network access layer which describes communication channel, its properties (electrical, transfer speed, full/half duplex), access methods, synchronization, data protection, etc. TCP/IP model is not concerned much about network access layer since mostly in praxis there are protocols standardized by ITU that already suit the ISO/OSI physical and link layers. For instance Ethernet, Token Ring, ATM, FDDI or HDLC.

Ethernet belongs among the most utilized and popular Internet protocols. Its transmission unit is called frame and consists of:

- Preamble (7 Bytes) – Allows clock synchronization.
- SFD (2 Bytes) – Start Frame Delimiter
- MAC Destination Address (6 Bytes)
- MAC Source Address (6 Bytes)
- Length/Type (2 Bytes)
- MAC Data and Pad (46 - 1500 Bytes) – Data of higher level protocols.
- FCS (4 Bytes) – Frame Check Sequence.

The most important element for measurement is MAC Data which contains headers and data of higher level protocols. The administrator might also be interested in MAC addresses and number of packets with bad Frame Check Sequence. For design of the probe parameter of interest is the minimal interval between arrivals of two subsequent frames on the 10-Gigabit Ethernet. It gives the minimum time that is left for processing of one frame.

Table 2.1: Intervals between arrivals of two consequent frames

	Packets/sec	**Min. Interval**
Ethernet IPv4	14.88 million	67ns
Ethernet IPv6	12.02 million	83ns

2.1.2 IP Network Protocol

Main task of the IP protocol is to deliver datagram from source node to the destination node. It does not establish any channel and instead every datagram contains source and destination IP address according to which are individually routed throughout the Internet by a net of cooperating routers.

There are two versions of IP protocol: IPv4 and IPv6. IPv4 use 32-bit IP addresses segmented in class. Wasteful use of these addresses enforced introduction of sub-netting and network translation. Despite this effort the space of IPv4 addresses will be exhausted and therefore IPv6 protocol with 128-bit addresses was introduced. Not only address length

changed but the whole header was redesigned to support new services most of all real time transfers (for details see IPv4 specification [25] and IPv6 [11]).

There are several header fields in IPv4 and IPv6 that might be interesting for monitoring:

- Version (4 bits) – IPv4 or IPv6.
- Type of Service (IPv4) or Traffic Class (IPv6) (1 Byte).
- Flow Label (20 bits) – Used for specifying special router handling from source to destination(s) for a sequence of packets.
- Protocol (IPv4) or Next Header (IPv6) (1 Bytes) – Specifies the next encapsulated protocol.
- Source and Destination Addresses (IPv4 - 8 Bytes; IPv6 - 32 Bytes).
- Payload Length (IPv6) (2 Bytes) – The length of data in the packet.
- Total Length (IPv4) (2 Bytes) – The length of datagram (header and data).

It is necessary to allocate sufficient resources (memory and performance) to handle both IPv4 and IPv6 datagrams. Moreover there are built-in extension of IP protocol to provide necessary Internet services, e.g. ICMP, IGMP. Monitoring of these protocols provides a lot of valuable information to the operator so they should be included as well.

2.1.3 Transfer Layer and TCP/UDP protocols

The task of the transfer layer is to determine the application the packet belongs to and to provide services the application demands. For instance TCP [26] guarantees reliable transmission and flow control over unreliable underlying layers. Some application sacrifice such features for the speed and simplicity (e.g., real-time) which is provided by UDP [24]. Both protocols have common fields called source and destination ports. There is a list of applications given by IANA that listen on known ports and provide their services. Monitoring of these fields gives an administrator the perspective what services are utilized in his network. There are another interesting fields worth of watching:

- Source and Destination Port (4 Bytes) – Determines the application.
- Sequence Number (TCP) (4 Bytes) – Used for in-order delivery.
- Control Bits (TCP) (1 Byte)– Used to establish, maintain and cancel the session.

2.1.4 Application Layer

In TCP/IP the Application Layer includes also data presentation and session management. Applications use its own data encoding although there are some exceptions that utilize standardized presentation protocols (e.g., SNMP use ASN.1). There are numerous application level protocols in TCP/IP, including Simple Mail Transfer Protocol (SMTP) and Post Office Protocol (POP) used for e-mail, Hyper Text Transfer Protocol (HTTP) used for the World-Wide-Web, and File Transfer Protocol (FTP). Most application level protocols are associated with one or more port numbers and can be tracked as mentioned in previous section. Content analysis of application data is computationally demanding but might be worth of it. It is possible to detect known patterns of worms, viruses or to prevent

spams. Data analysis is hampered by many obstacles. First of them are different encodings standards. Next when applications are using compression or encryption it is impossible to check the original content. Moreover the speed of the incoming data is usually very high to look for more then several patterns using an ordinary PC. Last but not least is the ethical issue. Monitoring at the level of IP and Transport Layer is considered to be ethical as it is necessary for maintenance of the network and incident localization. But scanning of users private data, even on the public media, is subject of many discussion and should be defined by the law.

2.1.5 Dynamic properties

The previous description is static in means that it describes the packet at one time interval. But the packet has also dynamic characteristics. The packet may change during its lifetime either because of transmission errors or because it has to. The second possibility is less obvious but happens at each network node all the time. For example when router or switch change destination physical address of next hop device or when the router decrements number of remaining hops or splits packet into several fragments etc. Monitoring must take this into account and consider possibilities how to compound different information from different location about the same traffic. For example the study how to derive a packet trajectory through the network is published in [13].

2.2 Types of Network Monitoring

There is a lot of ways of network traffic monitoring. It depends on network device what type of monitoring is supported and also on needs of network operator. Operator is usually interested in following network or network traffic characteristics:

- Network device management (utilization, fault detection)
- Spectrum of traffic mix
- Network services
- Attacks detection (DoS, dDoS, spoofing, worms, spams)
- Packet delay
- Billing and accounting, keep policy

According to the level of aggregation we can divide:

- Total aggregation
- Defined level of aggregation
- No aggregation - Packet traces

2.2.1 Simple Network Monitoring

Usually each network device has means to provide data about its utilization and configuration. Hardware based devices holds these values in counters and registers, software implementations as variables or files. A network operator is typically interested in total number of packets and bytes, dropped number of packets and bytes. These statistics are typically moving averages over relatively long time windows (e.g., 5 minutes), and serve as a coarse-grain indication of operational health of the network. The most common method of obtaining such measurements are through the appropriate SNMP MIBs architecture:

- SMI - Structure of Management Information - Data structure
- MIB - Management Information Base - Object definition
- SNMP - Simple Network Management Protocol

SMI uses presentation ISO standard ASN.1 for data types and syntax description and together with BER (Basic Encoding Rules) allow data to be exchanged independently on the architectures. MIB is a virtual storage of managed objects which are hierarchically ordered. When the management station wants to know the state of the managed device it sends SNMP messages with request which is processed by SNMP agent at the device. After retrieving the information the agent replies to the management station. Another type of communication between management station and managed device is based on trap conditions (events like failure, overflow, etc.). When the conditions are met an unsolicited SNMP message is sent to the management station. For large networks the amount of communication caused by SNMP would be very large. Thus the RMON (Remote Monitoring) architecture based on agents and network management station is used.

Concerning of the device utilization the previous technique is quite suitable. But it fails to give details about the composition of the traffic mix as it reports only the total amount of traffic received by the device [18] or to detect any attacks. To sum up this method of monitoring is sufficient for smaller networks where operator does not have to handle difficulties with network topology and where incident handling is not necessary.

2.2.2 Packet Capturing

In most cases, aggregate link statistics are not sufficient to distinguish short-term changes on the link (the attack might be averaged out in a 5 min window). The opposite solution is to sample and capture individual packets. Selected information about each packet is sent to the remote collector using for example PSAMP protocol [14]. It is possible to estimate how many bytes and packets belongs to each source/destination address and port or to slice and dice the data according to protocol number, MPLS obtaining different views on the traffic. Moreover it allows to detect various attacks. Suppose an operator sees too many packets with identical destination and port address and different sources then he can estimate that the traffic belongs to DoS attack.

Packet sampling is ideally suited to estimate the composition of the traffic (e.g., on a link) in terms of various attributes (source and destination address and port numbers, prefix, protocol number, type of service, etc.) Typically, unfiltered sampling would be used to obtain a coarse-grained view of the traffic on a link, say. Once the characteristics of an interesting subset of traffic (e.g., a service type, or a source address prefix corresponding to some customer) has been identified, the resolution can be refined by filtering out this

traffic, and by boosting the sampling rate correspondingly. In this way, the traffic can be examined and characterized ("sliced and diced") arbitrarily [14].

Spatial reconstruction of packet trajectory [13] is also considered by this method. It is based on hash function which is computed over a set of stable field of IP packet header and payload. Routers compute a hash over incoming packets and if it falls in certain interval the packet is sampled. Information from several routers discover the packet path.

2.2.3 Packet Inspection

Capturing and analyzing each packet is scope of method called deep packet inspection. It is a combination of classification techniques and payload checking. Classification allows the operator to define rules how to process the packet during the next phase of payload checking which is vital to reduce number of false alarms and to improve performance. For example classification allows to constrain the set of patterns that are relevant for packet. It makes no sense to look for patterns malicious for SMTP protocol in web traffic. As mentioned above the packets are checked whether they contain certain patterns. If so they cause an alert which is handled specifically according to user defined rules. Usually the alert is logged together with packet that caused it.

Deep packet inspection is utilized in so called Intrusion Detection Systems (IDS) and Intrusion Prevention Systems (IPS) together with firewalls. Firewalls filter incoming traffic so the packet inspection does not have to analyze all the traffic. The difference between IDS and IPS is that IPS can block malicious packets entering the network. For the same ability IDS have cooperate with firewall and to create a firewall rules which may come late after an attack. Most of currently running IDS are therefore passive and the alerts are left for manual solving (i.e., email to the local administrator).

Intrusion systems evolved during time very similar to firewalls as they shifted from stateless to stateful analysis. Stateful IDS/IPS is able to reassemble TCP stream and looks for the patterns seamless across the packet boundaries. On the contrary stateless analysis does not contain any records of the previous seen packets and generally is less powerful as the attacker can hide the malicious pattern in two consequent packets. On the other hand when dealing with the performance issues then the stateless is less resource consuming.

The rules for the packet inspection must be created by the administrator according to his experience. Alternatively there is an open database of known threats in a form of suitable for inspection systems. This de facto standard is set by program called Snort which is supported by a broad network community. Nowadays the database contains several thousands of rules and more comes every day along with newly discovered threats and vulnerabilities.

Unlike previous described methods deep packet inspection is focused on high security of the network based on the detection or blocking of malicious traffic.

2.2.4 Flow-based Measurement

The Internet is based on the concept of packets which travels across networks from its origin to its destination. There are no means to guarantee any bandwidth for packets transmission, i.e., to create a virtual channel with assigned network resources. But instead of channels we can observe that the corresponding packets create flows which are mutually mixed and fill the bandwidth(see Figure 2.2).

The aggregate information about flows are able to provide crucial information not only about volume but also about spectrum of the traffic mix and about behavior of individual

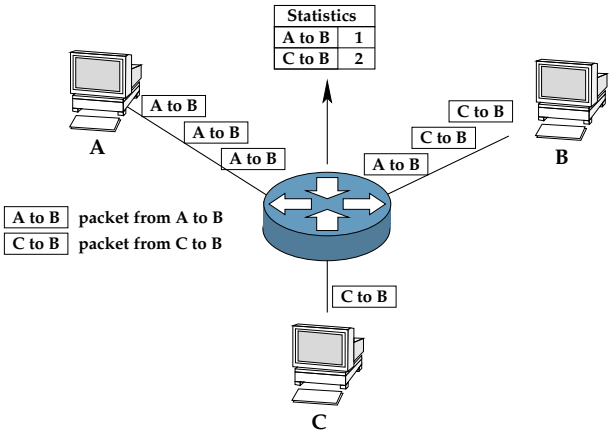


Figure 2.2: Packets of the same values in header belongs to the same flow and statistics can be performed upon them

entities on the network. Flow measurement is used in many application such as:

- *Accounting and billing* – Flow data are able to provide essential information for various accounting models. The flat-rate scheme can be exchanged for the usage-based accounting [5]. The Internet Service Provider(ISP) can offer a customer a flexible billing scheme based on for instance time-of-day, bandwidth usage, application usage, quality of service, and so on. The essential elements needed for accounting are the number of transferred packets and bytes and timestamps per flow.

  Note that current flow monitoring systems does not provide the reliability required by usage-based billing-systems as defined in [5] because of unreliable transport protocol.

- *Network planning* – Measurement results monitored over a long period of time allows to track and anticipate network growth and usage. Trend analysis allows to plan new network topologies, discover bottlenecks in the network and replace them with more powerful systems. Moreover it allows to optimize strategy of routing and peering. Decisions are made according to knowledge of flow duration, flow volume, burstiness, the distribution of used services and protocols, the amount of packets of a specific type, etc.

- *Traffic Engineering* – The goal of traffic engineering is to optimize network resource utilization and traffic performance [6]. The analysis relies on information about link utilization, load between specific network, nodes, number, size and entry/exit points of active flows and routing information [27]. These parameters can be successfully derived from the flow data and allows to optimize load balancing traffic across alternate paths or by forwarding traffic to a preferred route. Moreover it provides ISP peering partners with the ability to measure the volume and characteristics of traffic exchanged with each other and to modify their peering agreements.

- *Application and User monitoring and profiling* –

  Content and service providers are able to plan and allocate network and application resources (such as database server sizing and location) according to gained flow data. Flow data provides them with statistics about application usage over time and network.

The same what holds for applications can be applied to users. The ISP can monitor customers behavior in order to efficiently plan and allocate access, backbone, and application resources as well as to detect and resolve potential security and policy violations.

Storing and mining in the flow data enables ISP to create top lists and aggregates of their customers traffic, most used services and applications.

- *Network monitoring and Security analysis* – Flow monitoring can reveal potential network errors, attacks and intrusions. Significant differences in actual and in common traffic patterns are alarming indicators of for instance routing problems, DoS attacks, affected computers, etc. Using several sources of flow data can speed-up the problem localization and resolution. Moreover exchange of flow data between ISPs allows to discover and track incidents that cross ranges of local networks.

In comparison with per interface summaries and packet capturing, flow level statistics provides more fine-grained application data than SNMP method and less volume than capturing every packet. Moreover the level of detail can vary with different aggregation levels. The basic unit created by the measurement device is a *flow-record*. These records are usually post-processed and aggregated according to operators needs in a database like way. Aggregate statistics per Flow allows the operator to drill down into the traffic on a link, and obtain measurements that are of selectable granularity both in space and time.

Chapter 3

Standards of Flow Monitoring and Export

The definition of the flow can be found in [27]. Briefly it is a set of packets with common properties (referring to as key-fields) passing an observation point in the network during a certain time interval. Other studies contributes to this definition by adding up details. For example activity and directionality of flow is added in [8].

Especially the activity is very important parameter of flow in monitoring devices. A flow is active as long as observed packets that are meeting the flow specification are observed separated in time by less than a specified timeout value. As soon as the flow is not active it can be released from the memory and reported.

Claffy [8] defines a flow as unidirectional or bidirectional. While the connection-oriented TCP traffic generally exhibits bidirectionality, it often exhibits very significant asymmetries in the traffic profile of the two directions. Each TCP flow from A to B also generates a reverse flow from B to A, at the very least for small acknowledgment packets. On the other hand more general is unidirectional definition since UDP and also other protocols are in fact unidirectional. Besides unidirectional flows can be transformed into bidirectional flows during the analysis process.

The rest of the chapter briefly describes the idea of flow information acquisition, its export and processing. Further it focuses on individual protocols for flow information export.

3.1 Exporters and Collectors

Basic units of whole system are exporters (routers or probes, Figures 3.1, 3.2) . They are usually situated at an important locations of the network, near gateways or borders of networks.

By tapping the link they gather statistics and export them to the collector unit . Exporters are passive flow monitoring devices which identify packets according to source and destination IP addresses, ports, protocol and/or other fields. This way packets are assigned to the corresponding flow. The flow-record is kept in memory as long as the flow is active. Once the flow ends exporter wraps statistical data (flow-record) into flow information protocol and sends them to the collector. But it is not always easy for exporter to find out whether the flow has already ended. Unlike TCP flow, other protocol do not have start, end or reset flags to signal end of the session so other methods come into play. For example

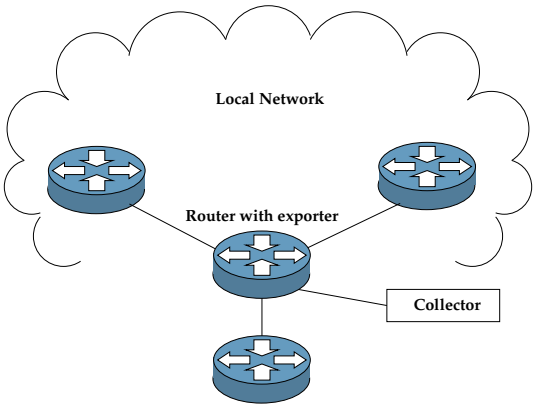


Figure 3.1: Standard flow monitoring system today at the border router

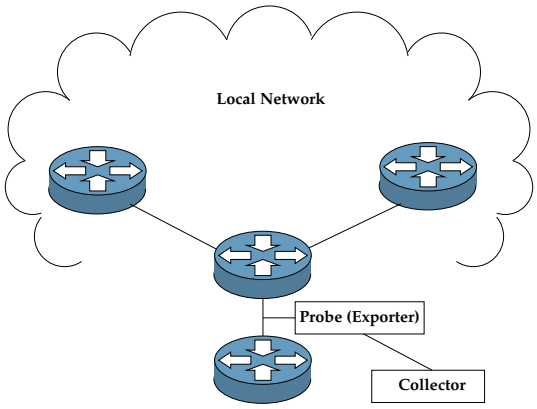


Figure 3.2: Solution with probe

measurement in time bins or inactive and active timeouts. These techniques are described in Chap. 4.

The exported flow-records are essential information basis for the traffic analysis in collector. They are either stored in raw format or entered in database. It depends on the way the operator wants to handle the data and what are expected analysis operations. Parameters as performance, quality and convenience during receiving, processing and post-processing must be considered, e.g., when receiving large amount of flow data entering them in database is more expensive than to store them in raw format but their analysis (aggregation, selection, etc.) would be more convenient.

Collector usually receives flow-records from several probes which allows operator to compare and match traffic originating from different locations in network. For example the administrator receives an email about the dDos originating from his network. The first thing he can do is to check whether he has seen the same attack at the border probes, i. e., huge amount of flows targeting one server in victim network. If so, he can track down the vulnerable servers that was used to cause DoS and further to track down the attacker who used this vulnerability. Using a local network probes he can estimate the approximate source of the attack and notice a local authority to track and solve incident further.

Collector side requires large storage fields because original flow-records are stored for weeks. During this period they are accessed and processed many times. Spectrum of traffic, attacks or top-lists are gained by selection and/or aggregation of stored data using a database queries or command line scripts. Popular representation of measured results is in the form of graphs with temporal X-axis.

For most of the operators the recent data are of the greatest importance as the incidents are reported maximally in two days since they had happened. While old flow-records are post-processed and aggregated to release storage space (Figure 3.3) [30].

3.2 NetFlow

NetFlow is Cisco proprietary protocol. It defines flow-record content, flow export format and related terminology. Improving monitoring abilities of network devices required changes in its definition and therefore several other versions of NetFlow were introduced. Most commonly used is NetFlow v5 and NetFlow v9. NetFlow v5 is very popular for its ease

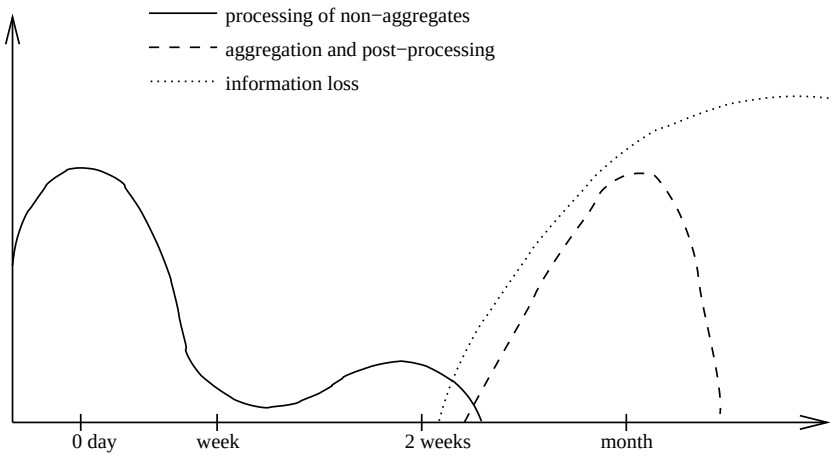


Figure 3.3: Lifetime of data in collector

of use. It has a fixed format of flow-record as well as fixed discriminating key-fields. The version 5 record is described in table 7 in Annex A. UDP stream is used to export data to the collector. Version 5 is supported by most of the exporters (simple hardware implementation) and collectors.

NetFlow v9 is much more developed and benefit from large experience in real environment of previous versions. Its concept is based on variable flow-records which significantly distinguish it from previous versions. Its variability is based on user-defined templates. Template is a collection of fields along with the description of their structure and semantics [9]. Template mechanism allows the user to select which fields he is interested in. Then the monitoring can be focused only on these fields which can lead to higher performance, memory and bandwidth savings when exporting smaller records to the collector. Another significant advantage of templates is extensibility in the sense that new fields can be added to the flow-record any time. Records extension does not imply new version of the exporting protocol any more. The templates contain vital structural information for the collectors to be able to process the data. It is thus important to resend the template descriptions every once in a while and before the first data transmission. Even if the collector does not understand the semantics of any field in the flow-record it is still able to work with the field because it knows its structure.

An exported packet consists of a header followed by one or more FlowSets. FlowSet is a generic term for a collection of records which have similar structure. There are three different types of FlowSets:

- Template FlowSet
- Data FlowSet
- Option FlowSet

An exported packet contains one or more FlowSets, and the three FlowSet types can be mixed within the same Export Packet (see Figure 3.4).

Packet Header consists of several fields but the most important are *Version* (version of flow-record format exported in this packet) and *Source ID* (Source ID field characterizes the Observation Domain. Collectors should use the combination of the source IP address

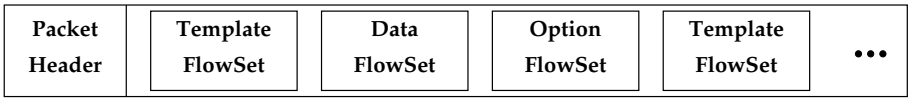


Figure 3.4: Format of NetFlow version 9 packet

and the Source ID field to separate different export streams originating from the same Exporter).

The template FlowSet contains template definitions. Each template is marked with unique ID and a list of field definitions – a pair of field type and its length (an example is given in table 3.2).

Table 3.1: Definition of items in flow-record

TYPE IDENTIFIER	LENGTH(Bytes)
8 (IPV4_SRC_ADDR)	4
12 (IPV4_DST_ADDR)	4
1 (IN_BYTES)	4

Data FlowSet consists of flow-records each assigned to a given template by the template ID. Option FlowSet is used to advertise collector settings of exporter.

NetFlow version 9 was designed to be independent on the transport protocol so the congestion aware protocols might be used but for practical reasons such as efficiency, performance and simple implementation UDP datagrams are utilized.

3.3 IPFIX

Unfortunately NetFlow is Cisco proprietary. Therefore Internet Engineering Task Force (IETF) introduced open standard in the form of RFC document for flow information export. The protocol is called IPFIX and allows all network devices capable of collecting flow-oriented statistics to export it in a unified way.

The IETF workgroup proposed several drafts that contributes to flow measurement and export. Some of them are listed together with their status to give the reader possibility to look for closer information if interested. Next paragraphs try to sum up information collected from these drafts to create an information basis that can be used during the design phase of the flow probe.

1. IPFIX Requirements (RFC 3917)
2. Architecture for IP Flow Information Export (IESG Evaluation)
3. Specification of the IPFIX Protocol for the Exchange of IP Traffic Flow Information (RFC editor Queue)
4. Information Model for IP Flow Information Export (IESG Evaluation)
5. IPFIX Applicability (Waiting for IESG Evaluation)

IPFIX Requirements [27] and Architecture [28] give general details about flow monitoring system(terminology, concepts), assign tasks to particular processes (metering process, exporting process, collecting process) as well as discuss security issues.

The IPFIX protocol [10] definition requires congestion aware transport protocol to export flow-records from exporter to collector which concerns of packet loss, packet retransmission, etc. Suggested protocols are SCTP or TCP. UDP protocol can be used but it is optional. Using congestion aware and secure channel allows to solve some of the security issues (eavesdropping, packet forgery) and allows its usage for accounting and billing applications. IPFIX protocol much like NetFlow v9 uses templates containing pairs of type and length to specify which fields are present in data records. Different data records may be transmitted simply by sending new templates specifying the pairs (type and length) for the new data format.

Information Model for IP Flow Information Export [12] defines a large number of standard Information Elements which provide the necessary type information for templates. The use of standard elements enables interoperability between different vendors implementations as well as it helps to derive requirements on the hardware resources (64-bit counters, various timestamp precisions, etc.).

IPFIX Applicability [31] draft shows how applications can use IPFIX. It describes the relevant information elements and shows opportunities and limitations of the protocol. The document furthermore describes relations of the IPFIX framework to other architectures and frameworks.

Chapter 4

Heuristics and Algorithms of Flow Monitoring

The flow monitoring consists of packet sampling, timestamping, packet header parsing, and maintaining flow records. Several algorithms and heuristics were defined to assist during the monitoring process.

4.1 Flow maintenance

The maintenance of flow records includes creating new records, updating existing ones, detecting flow expiration, passing flow records to the exporting process, and deleting flow records. The measurement process has therefore means how to control all its tasks, some of them are simple but some of them require closer explanation which is given in following paragraphs.

When the first packet of the flow is seen by the monitoring process the flow record has to be created. The address of the memory location for the flow record is usually computed from the key-fields of the packet. Subsequent packets of the same flow updates the flow record.

The monitor also keeps an extra state for every flow in the memory to determine the time when to release the flow record to the exporting process. There are several conditions under which the flow record should be expired:

- *Inactive timeout* – Flow records are periodically checked whether they have received packet recently. If the interval since the last received packet and current time is greater than the Inactive timeout then the flow is considered as inactive and reported.
- *Active timeout* – This criterion guarantees that the long lasting flows are reported. For instance imagine the communication of two SMTP servers whose communication probably never ends. Then there would be no reported record. To prevent it the flows are checked for their activity and reported if exceed active timeout.
- *Explicit notice* – There are protocols such as TCP that explicitly advertise their start and end of flow. These flows are considered inactive when the end packet is transmitted.
- *Resources* – Measurement process can expire flows when it experiences the lack of resources.

Control mechanisms are usually implemented in different ways according to the required designation of the device. For instance implementation of software based probe would significantly differ from the one implemented directly in ASIC.

4.2 Enhancements and advanced techniques

During past years new techniques or heuristics addressing the flow measurement were published each contributing in one of the following fields:

- Control
- Resource protection
- Quality of results
- Performance improvements

4.2.1 Expiration control

In work [17] is stated that most traffic analysis tools divide the traffic stream into fixed intervals of time. Therefore instead of expiring flows upon the conditions mentioned in the beginning of this chapter, the measurement in time bins is proposed.

Time bins measurement gathers statistics for a fixed length interval while no flows are expired. At the end of each interval all aggregated data are expired at once. Of course some flows spread across several bins and therefore are recognized by collector as several consequent flows . For example when the flow starts at the end of the bin it is immediately exported and monitored again in the next bin. The situation is explained in Fig. 4.1.

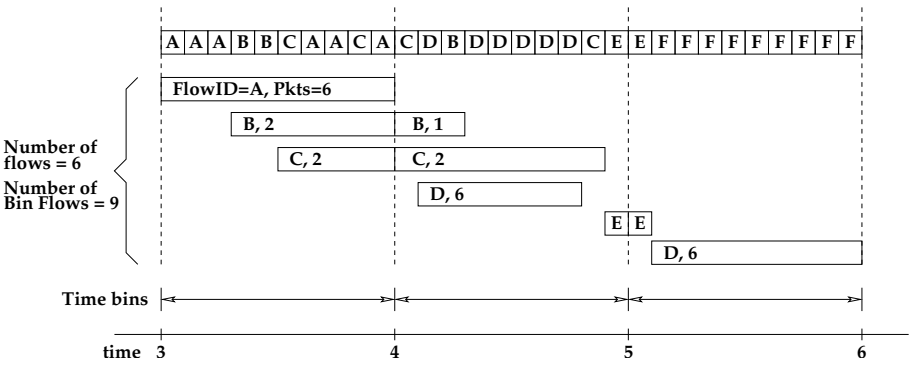


Figure 4.1: All flows are exported after a fixed interval (some of them are splitted)

This kind of expiration control is suitable for analysis tools that work with post-processed, aggregate results while the precise timestamping and primary flow records are necessary for applications such as security analysis, one way delay measurement, etc. Advantage of timebin approach is that it is not necessary to detect which flows are expired because all of them are reported at the end of the measurement interval.

In most flow-monitoring devices, classical approach (described in Sec. 4.1) of flow expiration control is implemented. Also IPFIX and NetFlow protocols requires flow-oriented expiration.

4.2.2 Protection

Flow monitoring systems are vulnerable to various attacks (dDos, DoS attacks, smurfs, port scans for instance). It is because the state is kept for all simultaneous flow present at the network. Attacking-packets usually do not belong to one flow but each creates a new one. Therefore flow systems are overwhelmed and unable to give appropriate and unbiased information about those attacks. Several techniques were published in last decade to protect monitoring elements. They are usually based on some kind of sampling or aggregation.

1. *Input Sampling* – Sampling of incoming packets is the easiest way how to guarantee minimal time between two consequent packets, thus to decrease the load of the processing engine. It helps also to decrease number of total flows. For example sampling one to ten reduce number of new flows approximately ten times. On the other hand sampling makes it difficult to estimate exact number of flows in original traffic (Fig. 4.2) [17].

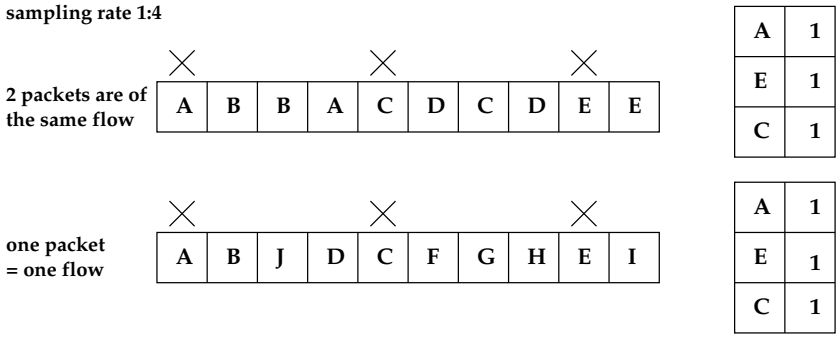


Figure 4.2: Wrong estimate how many flows are present

2. *Output Sampling* – Rate of expired flow records is strongly dependent on the traffic mix. Output sampling can keep transfer of flow records to the exporting process under certain level and prevent congestion at the exporter. Next stage of output sampling can be performed in exporter itself to lower the load on the link and on the collector.

3. *Sample and Hold* – This method is quite similar to input sampling but with following twist. As with ordinary sampling, each packet is sampled with a probability. If a packet is chosen and the flow it belongs to is not in the flow memory, a new item is created. However, after an item is created for a flow, unlike in Input sampling, every subsequent packet belonging to the flow updates the item as shown in Fig. 4.3 [18]. This method is able to significantly decrease number of new flows which can come handy during attacks while long lasting flows are not influenced because some of their packets always make it in and then are monitored with no sampling.

4. *Multi-stage accounting filters* – This method gives similar results as sample and hold. It utilizes small amount of memory to measure how many bytes came for each flow and multiple hash functions that address this memory by computing a hashes from the flow ID. When packet comes its length is accumulated to all items addressed by the hashes. If all currently addressed items have greater value than specified threshold then the flow is considered large enough to be monitored [18].

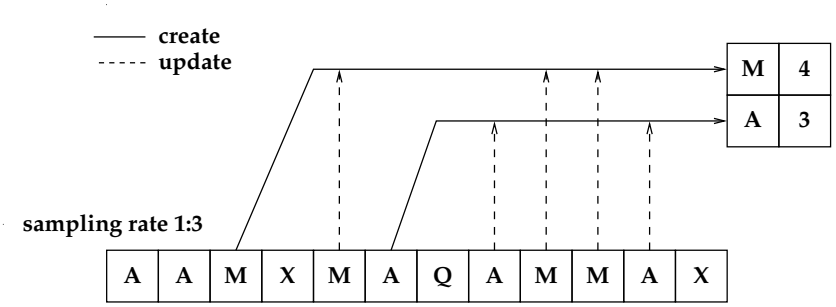


Figure 4.3: Once the entry has been created record is updated for every belonging packet

5. *Adaptive Input Sampling* – A static sampling rate is either suboptimal at low traffic volumes or can cause resource consumption (memory, bandwidth) or difficulties at high traffic volumes. Adaptive Input Sampling holds outgoing export within fixed resource consumption limits while using the optimal sampling rate for all traffic mixes [17]. The exporter is prevented from being overwhelmed by aggressive export rate during peaks or attacks by decreasing the sampling rate and when the critical traffic ends than the sampling rate is increased for example up to $1:1$ which means sample every packet.

   The adaptive sampling shows promising results in test scenarios. Unfortunately there are no implementations (not known to the author) of monitoring devices that would allow to configure and set adaptive sampling. Mostly it is because of uncertainty how to set and control it. The interpretation of exported data is also subject of discussion and therefore not supported by most of the collectors.

6. *Pre-aggregation* – Most of the sampling methods have the disadvantage of loosing information randomly. By a specified filter placed in flow probe we can choose according to which fields in packet header we want to aggregate thus selecting what type of information we are interested in and what we are willing to loose.

   For example let us imagine denial of service attack targeting one server and one port. If the flow measurement is configured to monitor according to destination port and IP address then this attack does not create millions of single packet flows but is aggregated as one huge flow (see Fig. 4.4). However pre-aggregated records miss a lot of essential information present in raw flow records.

4.2.3 Indexing and Addressing

When the packet comes, belonging flow-record must be addressed to update the statistics. The direct model of addressing (the whole key is an address in the memory) is not possible because the key has usually around 300 bits. That is why the hash-based addressing is used, i.e., the memory location is directly addressed by hash of key-fields. It is not always ideal and usually suffers from a lot of collisions which are handled by replacing the old records by the new ones. In pathological situation packets of two different flows which maps to the same spot replace instantly each other. It leads to no aggregation what so ever and causes problems in subsequent systems (e.g., congestion of exporter, collector, inaccuracy of monitoring). More efficient ways of addressing comes into play.

The flow identifier consists besides others from IP addresses and therefore it would be

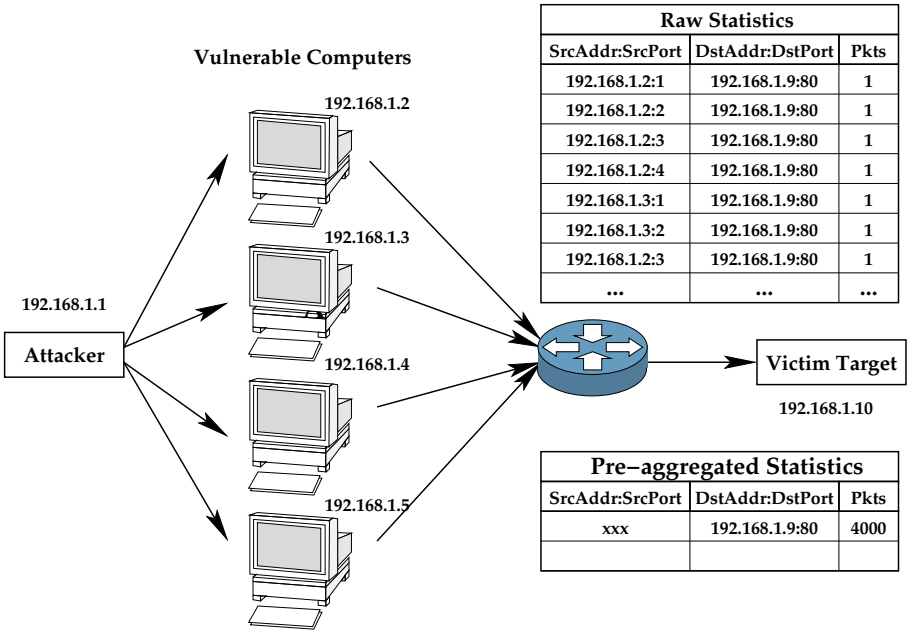


Figure 4.4: Raw versus pre-aggregated statistics during DoS

possible to use trie structure to look up the record. Unfortunately the time to look up the record is limited as well as the access to the memory is expensive. Therefore modifications of hash-based addressing are proposed and implemented. In [23] the search in the flow cache is once again based on the computation of a hash function on the flow identifier. The hash function result (or a masked subset of its bits) is used to index a vector of pointers to the lists of records, as shown in Fig. 4.5.

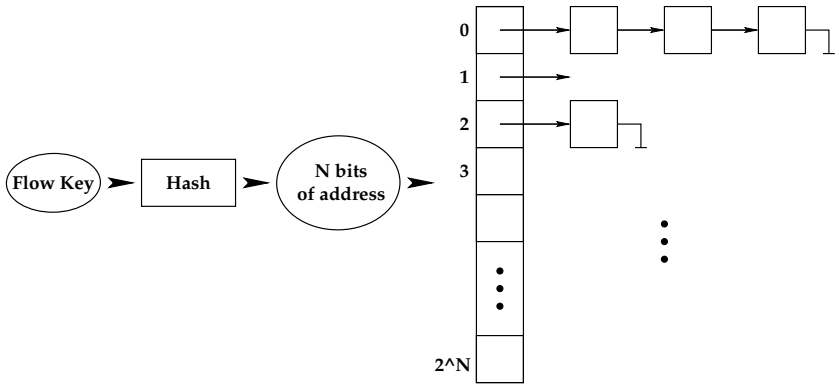


Figure 4.5: Hash and search addressing

Practical implementations of hash and search usually require two restrictions. Fixed length of the search list and instead of flow-records only pointers to the flow-records in the lists itself. There are several reasons to follow these rules. For instance fixed length of the list is required because of dynamic memory allocation in the PC can be be costly as well as hardware implementation could not benefit from the burst mode access to the memories. It would be necessary to follow pointers binding the list to collect all items in the list. Second constraint for pointers in the list is necessary to efficiently utilize the search

memory. Instead of whole record which is 64 bytes large it is better to hold 4 bytes pointer and 4 bytes additional identification. The locality is again better and also allows to split tasks of finding the location of the flow-record and processing of the flow-record.

4.2.4 Anonymization

Flow data posses a lot of information about users behavior, e.g., when do they browse the web what pages do they visit, etc. Moreover flow data might compromise hidden aspects of the network which could help attacker to launch an attack. Anonymization provides techniques to alter IP address and/or ports to hide most of the sensitive data. Anonymization is necessary when the flow data are publicly available such as on-line statistics, publications or when there is a suspicion of wiretapping the export. Several types of anonymization were proposed to support specific post-processing of flow data.

First set of algorithms consists of hashing and encryption algorithms (MD5, SHA, AES and DES). Applied to the selected fields they provide constant one-to-one mapping between original and anonymized items. These techniques are suitable for flow records upon which no further analysis is required. It is because they do not preserve original relative distribution of values in the traffic. To explain it suppose a DoS attack where the original traffic shows incrementing source IP address in certain range in subsequent packets. Anonymization of the traffic causes uniform distribution of the incrementing addresses across the whole state space with absolutely no relationship between each other. Thus the subsequent analysis would fail to detect the attack in anonymized data.

It is, therefore, highly desirable for the address anonymization to be prefix preserving (second set of algorithms). That is, if two original IP addresses share a k-bit prefix, their anonymized mappings will also share a k-bit prefix. One approach to such prefix preserving anonymization is adopted in TCPdpriv [22]. Another prefix-preserving anonymization is proposed in [19] utilizing a bit flipping anonymization tree where each bit of IP address is either flipped or kept according to the value in the tree (see Fig. 4.6).

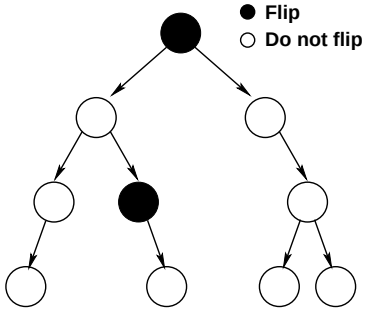


Figure 4.6: Example of anonymization tree for four bits

Prefix-preserving anonymization allows the flow analysis to process anonymized data and look for characteristics of the original traffic. On the other hand prefix-preserving anonymization is more sensitive to cryptoanalytical attacks and is more resource demanding.

4.2.5 Performance improvement using Temporal locality

Apparently, the speed of links grows faster than the CPU power and memory speed. As a result, memory access becomes a bottleneck in applications and devices that need to store

and update context information for every received packet. One of such applications is the flow metering process itself.

Knowledge of network traffic characteristics can improve throughput of metering process whose performance is often limited by slow access to memories. We suggest that design of such application in hardware could use a hierarchical system of memories in a similar way as the processor cache is used by the PC architecture. The success of such an approach depends on a deep understanding of the dynamical characteristics of the incoming data. In the case of network traffic the most important properties are temporal and spatial correlations (locality).

Flow metering process usually monitors wider network thus the spatial locality of the traffic seen by the process is not the direction that could lead to significant optimizations. On the other hand it is worth to explore temporal locality which means that if certain packet was seen in recent past it is highly probable that it will be seen in close future. To this end, we have performed measurements described in Sec. 6.2 which shows promising results.

4.2.6 Variable Flow-record

So far, implementations of flow probes have fixed structure of flow record. It means that there is no way how to reconfigure the flow record if the administrator is interested in another information (e. g. different fields of packet header). Although this feature is already supported by latter exporting protocols which use templates to define the structure and semantics of exported information.

We suggest to implement the configuration scheme using XML language. The administrator has a choice which fields to monitor. Implementation details are discussed in Sec. 5.6.

Chapter 5

System Architecture

The probe is supposed to be based on commodity PC running Linux OS with network acceleration card with 10-Gigabit interface. The card accelerates the time critical parts of the flow monitoring process. While the PC is responsible for post-aggregation and the export of the collected flow statistics.

The chapter is dedicated to the description of platform and implementation details of monitoring process suitable for FPGA. After that software architecture is outlined together with suggestions for aggregation process implemented in software. Each stage, either monitoring or aggregation process, requires implementation some of the procedures described in Chap. 4. Various algorithms are suggested and analyzed to select the best solution for the given procedure and platform.

5.1 Platform

The target platform of the firmware architecture are ten gigabit card XFPro and COMBO6X mother card. Both are described in following paragraphs.

The concept of COMBO acceleration cards is based on idea of hardware-software co-design. Hardware acceleration card is plugged in system bus of the host PC. Time-critical parts are implemented in firmware of FPGA while software part performs more general but computationally less demanding operations above preprocessed data coming from the card. The goal of the COMBO family is to give developers a possibility to work with "open hardware" and use it in the same way as open-source software [3]. Originally the cards were developed for a design of a multi-gigabit IPv6 and IPv4 PC-based router as a part of CESNET [1] research activity. Later on it showed up very useful to use them in broader variety of applications as their hardware resources provide a great computational power.

The heart of COMBO cards consists of one or more FPGA (Field Programmable Gate Array) chips, memories and other necessary components (power supply, IO chips, connectors, etc.). Due to the flexibility of FPGA chips, the functionality of COMBO cards can be easily (and quickly - within just several milliseconds) changed by downloading a new design into the FPGA. This way the same cards can be used for a wide array of research and development projects. Currently the most advanced and powerful cards are COMBO6X with XFPro. Their design was adapted to suit applications working with 10-Gigabit Ethernet. Both of them are described in following paragraphs.

COMBO6X (see Fig. 5.1) replaced older version of COMBO6 card to offer faster chips, larger memory capacity and compliance with new PCI architectures (PCI-X, PCI Express).

It consists of two Xilinx Virtex-II Pro FPGA chips, Ternary CAM, dynamic and static memories. Various add-on cards can be used with COMBO6X card.

Figure 5.1: Front side of COMBO6X card

COMBO6X is equipped with:

- Virtex II PRO - XC2VP50 (the series up to XC2V70 can be used)
- Virtex II PRO - XC2VP4 with PCI core (alternatively PCI-X)
- CAM - 2Mb ternary CYNSE70064A
- 3x SSRAM - 2MB 512K36
- EEPROM - 93S66
- PCI connector
- Connector for add-on card
- Extension/test connector
- DRAM connector for PC DDR up 2GB

Besides card for PCI-X there is a card for Express PCI as well. By supporting new bus standards – PCI-X and Express PCI – COMBO cards open new design possibilities for network devices. Hence the PCI bus is no more the bottleneck of the system and it is easier to utilize the host PC processor along with COMBO hardware. Specialized chips for PCI communication were replaced by pre-programmed modules directly in the XC2VP4 FPGA or XC2VP20 (Express PCI). Parallel buses were replaced by serial communication lines in order to prune away issues with feedthrough noise, clock distribution and lack of external pins. Serial links are supported by Xilinx Virtex-II Pro via fast serial RocketIO circuits inside FPGA.

Mother cards are ready to host variety of add-on cards. Their mutual interconnection is possible via RocketIO interfaces or ordinary parallel bus connector with about one hundred available wires. The add-on cards are equipped with different network interfaces such as four port gigabit Ethernet, OC-48 or 10 GE interfaces.

The add-on COMBO-2XFPRO was designed to support two 10-Gigabit network interfaces. Thus it can work as a repeater and when inserted in a line it is fully transparent for all network traffic (see Fig. 5.2).

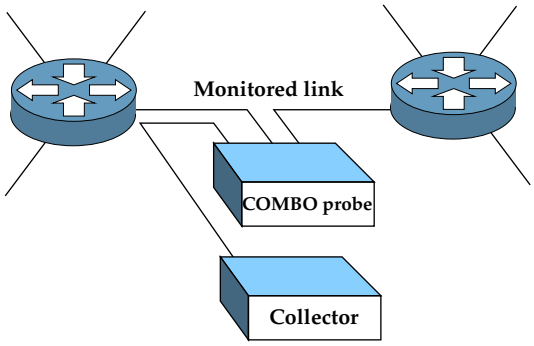


Figure 5.2: COMBO probe tapping the link

COMBO-2XFPRO is equipped with:

- Virtex II PRO - XC2VP20 (the series up to XC2V30 can be used)
- 2x SSRAM - 2MB 512K36
- EEPROM - 93S66
- 2x XFP cages
- Connector to the mother card

There is an hardware abstract layer called NetCOPE [29] for both cards, COMBO6X and XFPro. The NetCOPE provides standard interfaces to receive and transmit packets, it also allows to access available memories or to utilize communication connector between cards. Another already implemented block is FlowContext [21] that provides an access to the records in the memory. Due to these prefabricated blocks time to develop application firmware is shortened.

5.2 Architecture Overview

The system architecture is divided into two parts - hardware and software. The hardware part is intended to process and aggregate incoming packets at the wire speed. Whereas the software part post-processes records transfered from the card and exports them to the collector. The Fig. 5.3 shows the simplified architecture of the whole probe together with estimated input transfer speeds.

The first stage of probe in the FPGA decelerates incoming ten gigabit stream by aggregation packet information into flow-records. Before the flow-record is created or updated, incoming packet is processed at network layer L1, L2 and L3 to verify CRC, extract information about IP addresses, ports, protocol, length of the packet and other fields (complete list can be found in [27]).

Second stage aggregation implemented in software creates new or updates corresponding existing flow-record by the data acquired from the transfered flow-records from hardware.

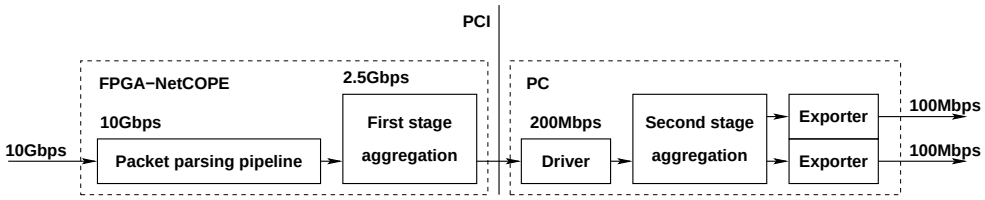


Figure 5.3: System architecture

The main reason for the second stage aggregation is to eliminate number of fragmented-flow, flows that were expired because of other reasons than timeouts (collisions, lack of memory). User-space application then wraps the records in datagrams and sends them to a collector using an ordinary network card (see Fig. 5.2).

5.3 Firmware Architecture

The probe has a ten gigabit network interface. All packets that are seen on this interface must be received (or sampled) and processed at the wire speed. The processing includes parsing of the packet, extracting available information, indexing the flow memory and aggregating information into the flow-records. The connection of the processing chain is shown on Fig. 5.4.

The task of the packet parsing and extracting the information out of the headers is left on the FPGA of XFPro card while aggregation process itself is suited for FPGA on COMBO6X.

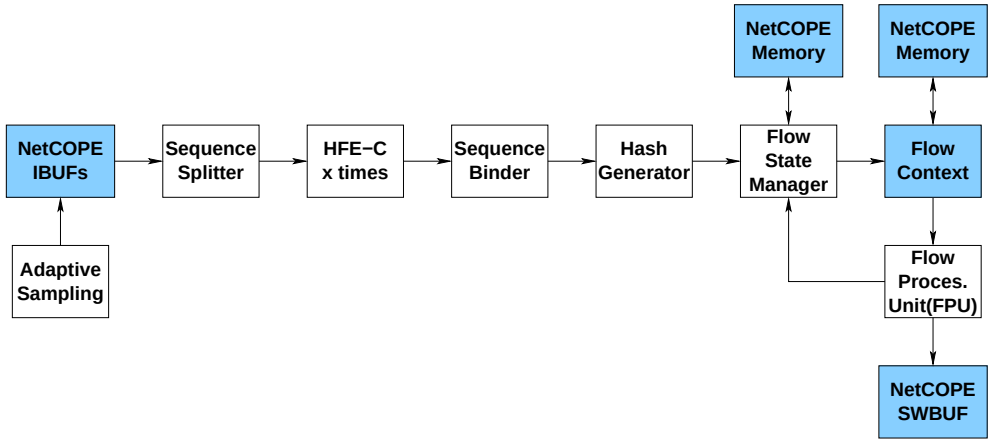


Figure 5.4: Firmware block structure

The application firmware utilizes NetCOPE network interface to receive packets. The interface provides packets at Layer 2 which means that received packets are already checked for the correct CRC, correct SFD and minimum and/or maximum transfer length. Each packet is also assigned with a 48-bit timestamp which resolution is 8 nanoseconds and range is 26 days. Very high resolution is used to assign each packet with unique timestamp so they can be correctly ordered in time. The source of timestamps could be either a GPS unit which guarantees a high precision of the timestamp or ordinary free running counter. In both cases the timestamp is converted to UTC time in software. The NetCOPE provides

interface to sample received packets. The sampling function though must be implemented in application itself which brings greater flexibility to the sampling process itself.

5.3.1 Adaptive Sampling Unit

Packet sampling at the input interface allows to decrease the intensity of the incoming traffic and thus decrease the load on the subsequent systems (probe, exporter, collector). It is the most desirable type of sampling as it preserves most of the information about the traffic in comparison with for example flow sampling. Unfortunately setting the fixed sampling rate wastes performance and capacity of the probe. Therefore it is better to use sampling which adapts its rate. Such type of sampling guarantees optimal resource utilization through various traffic mix.

Proposed Adaptive Sampling Unit has two inputs according to which the adaptation takes place.

1. Packet rate
2. Memory utilization

The adaptation function is implemented as a look-up table. Each row contains lower and upper boundary for every parameter and the corresponding sampling rate. The unit works as follows:

- The table is initialized by software.
- Row-pointer is set to zero, which means that row number zero is currently addressed.
- Input parameters are compared with lower and upper boundaries at the current row.
- If parameters exceeds lower/upper boundary in the row then the row-pointer is decremented/incremented and thus new sampling rate is selected. Go to previous point.

Note that it is up to user how to initialize the table. But it is recommended that he keeps certain hysteresis when setting the boundaries. So the row-pointer remain steady for longer time after each change (see Fig. 5.5).

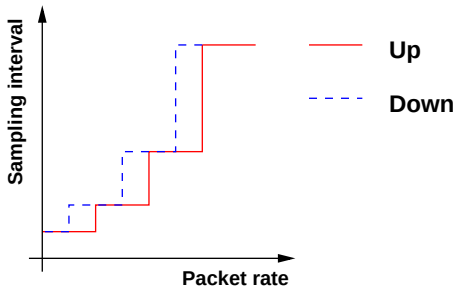


Figure 5.5: Hysteresis of adaptive sampling

5.3.2 Packet Parsing

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 (unified structure for all kind of IP packets) which contains data for the monitoring process. The example of Unified Header is in the Annex B. The HFE unit is implemented using Handel-C (HFE implemented in Handel-C is further referred as HFE-C). Handel-C is a modification of standard C language specialized for description of parallel computing structures. HFE-C compared to the standard has several advantages. For example HFE-C does not have an overhead of additional hardware and there are also no wait cycles during jumps. On the other hand the functionality of HFE-C is hard-coded and cannot be uploaded any time as it requires to reload the whole firmware of FPGA. The Handel-C implementation has also advantages when compared to the VHDL finite state machine. One of them is its abstraction of hardware structures, the programmer does not have to know anything about FPGA, at the same time it is nearly as efficient as the VHDL.

We have done comparisons of HFE implemented as processor and in HANDEL-C. Results shows that Handel-C is indeed right choice for effective packet header parsing (see Tab. 5.3.2).

Table 5.1: Comparison of HFE-C and processor HFE

Implementation	**Frequency**	**Throughput**	**Resources**
HFE-C	125 MHz	1306 Mbps	572 Slices
HFE-C	100 MHz	1044 Mbps	550 Slices
processor HFE	100 MHz	782 Mbps	510 Slices + 2 BRAMs

The table shows that the performance of the HFE-C running at 125 MHz is sufficient to process more than one gigabit traffic. Hence ten gigabit incoming stream is distributed among eight of these parsers. Each HFE-C can process one word of packet every clock. Unfortunately different lengths of packets causes different execution times which scrambles ordering of the packets. The ordering must be remained so it does not cause race conditions in Flow Processing Unit during updates of flow-records. Therefore all packets are marked with sequence numbers before they are dispatched to HFE-Cs. After HFE-Cs are done with their processing, they create unified headers which are ordered again by the sequence number into one stream. Fig. 5.6 displays the situation.

5.3.3 Hash Generator

Fields that determines the flow (known as key-fields) are subject of the hash function. Its result is the address to the memory of flow-records. Collisions caused by hash (two flows map to the same memory location) are handled in the first aggregation stage where the old record is exported to the software and new record is created for the new packet. Simulations show that good hash function and sufficient memory capacity will keep the collision rate reasonably low (see Sec. 6.3 for modeling of the collisions).

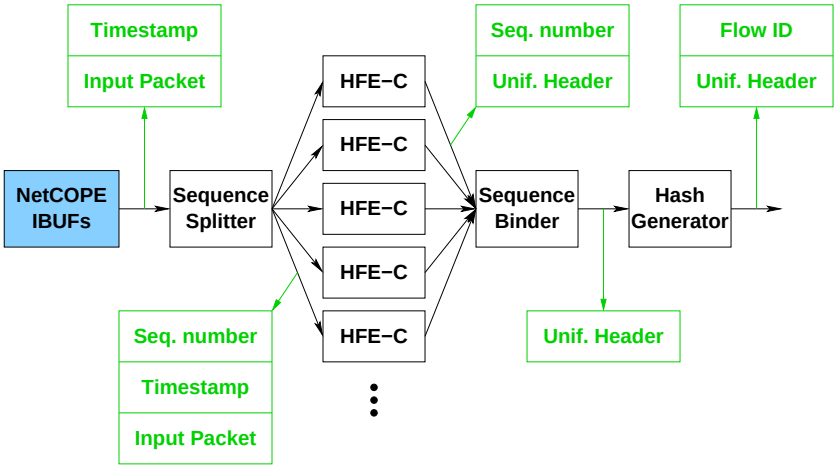


Figure 5.6: Sketch of processing of incoming traffic by several HFE-C

5.3.4 Flow State Manager

Flow State Manager is intended for keeping state of all flows in the memory. State of the flow means an information about its lifetime in this context. 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 administrator) then the flow is released. Please note that in all following algorithms timestamp can be of arbitrary bit-length and has nothing to do with the timestamp assigned to the packet at the input interface.

Several possible algorithms are able solve this task. First of them 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 list pointers. The idea of ordering is simple. New or updated flows are rebounded to the top of the list and their timestamps are updated. This way the least recently used (inactive) ones remain at the tail. Then it is easy to identify inactive flows.

Next of them can be in short described as *Field of sequentially decremented counters*. It works as follows:

1. Every incoming packet causes setting the counter for the given flow to the maximal value.
2. All non-empty counters are periodically checked and decremented.
3. If the value of counter reaches one then the flow is considered to be expired.
4. 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 countdown of counters.

Third algorithm stores timestamps of last seen packet in the memory which is again periodically checked whether the interval between the last seen packet is longer than inactive

timeout. One bit of each word is allocated to denote emptiness of the item. It is necessary to store only significant bits of the timestamp counter because of limited memory resources. For example, only four bits are allocated for the timestamp and the inactive timeout is set to fifteen seconds. In this case four bits that represents the range of 16 seconds are taken from the timestamp counter, stored in memory and used to compute inactive timeout criterion afterwards. If the inactive timeout is one second then lower bits of the timestamp counter are utilized. Every time the inactive timeout is changed then all stored timestamps must be rescaled so that they fit to the range of current timeout. To suppress this negative effect we suggest to implement the timestamp counter with the same bit-length as the memory word. The inactive timeout increases/decreases the clock rate for timestamp counter. The shorter the inactive timeout the faster the clock cycle and the sooner the flow is expired. The scheme of third algorithm is described on the Fig. 5.7.

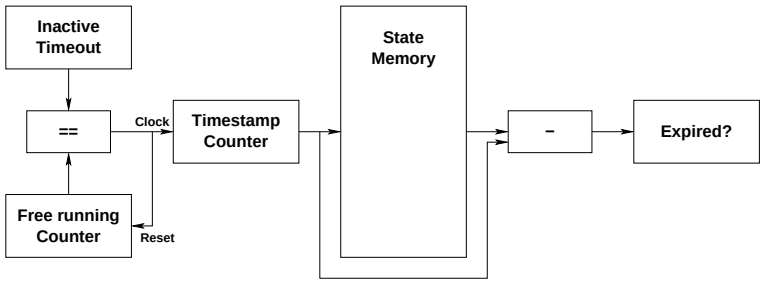


Figure 5.7: Scheme of Algorithm with speed variable timestamp counter

To sum up, all methods have its pros and cons. For example, bidirectional list requires an extra memory for list pointers or several memory access to rebind the item. On the other hand the only record which has to be checked for the inactivity is the one at the tail of the list. Bidirectional list seems to perform better in software than in hardware. Second algorithm has a lot of advantages. It is memory efficient, all the bits of the word are maximally utilized and so this setup has with same memory requirements the largest resolution or range of all algorithms. It has one disadvantage which is caused by the requirement to read and write the value that is checked and decremented. This disadvantage is solved by the last algorithm where the timestamp is only read out of the memory and only if the the inactive timeout criterion is fulfilled the empty bit is written. Unfortunately it is necessary to have an extra bit to denote the emptiness of the item. Following Tab. 5.3.4 sums up pros and cons of described algorithms.

Table 5.2: Comparison of flow state tracking algorithms

Implementation	**Quality**	**Mem. Access**	**Mem. Efficiency**
List	Excellent	Poor	Poor
Counters	Very Good	Good	Excellent
Timestamps	Good	Excellent	Good

Second and third algorithms were identified as suitable for FPGA implementation of Flow State Manager. The timestamp algorithm is the best solution with usage of external memories.

5.3.5 Flow Processing Unit

Main task of Flow Processing Unit is to aggregate information about packet into the flow-records. To this end it is connected to the FlowContext which is described in Sec. 5.1. 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 Unit 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 must work with the same flow-record in parallel.

The Flow Processing Unit performs also other tasks besides the aggregation. Successful establishment or update of the flow-record is dependent on the following sequence of operations:

1. Flow-record Control
   - Check if the flow-record is not empty
2. Unified Header Control
   - Check if the Release command is set
   - Check if the unified header is not empty
3. Collision Control
   - Compare all key-fields of UH and flow-record
4. Additional Controls
   - Active timeout control
   - TCP bits control
   - Overflow controls
5. Record operation establishment, update, release or no operation
   - Establishment of a new flow-record, mark flow-record as non-empty
   - Update of existing flow-record
   - Swap of existing flow-records
   - Release of existing flow-record, mark flow-record as empty

The flowchart on Fig. 5.8 defines the plan for flow-record processing. The update of the flow-record consists of repeating sequence of following primitive operations:

1. Read data from Header and Context Memory
2. Process the data
3. Store result back to the Context Memory

After the flow-record expires it is subject of optional sampling at the flow level. It can also be anonymized using a CRC hash function which can be easily implemented in hardware. Final flow-record is submitted to the NetCOPE software output buffer where it waits to be transferred via DMA to the kernel space of the operating system.

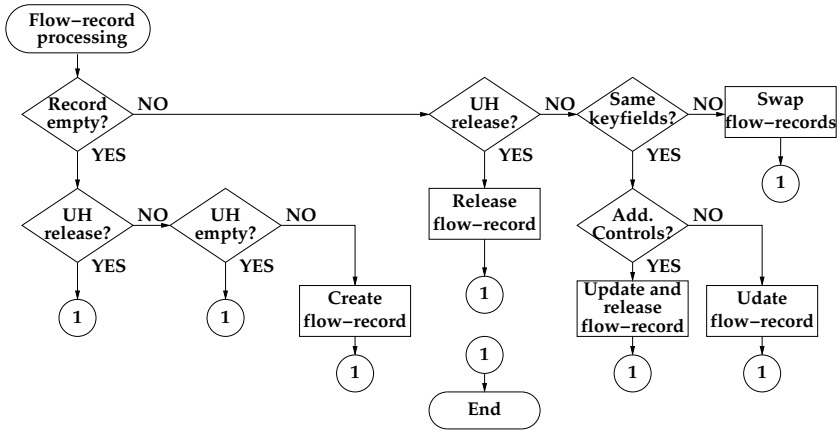


Figure 5.8: Flowchart of Flow Processing Unit

5.4 Software Architecture

The software of the probe consists of several programs (Fig. 5.9) – kernel driver, common flow library, second metering process, flow exporters exporters, programs for configuration.

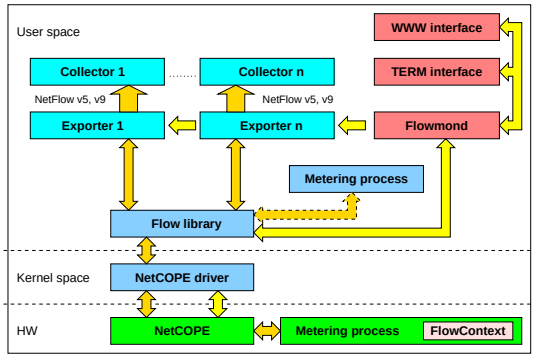


Figure 5.9: Layout of software with common second metering process

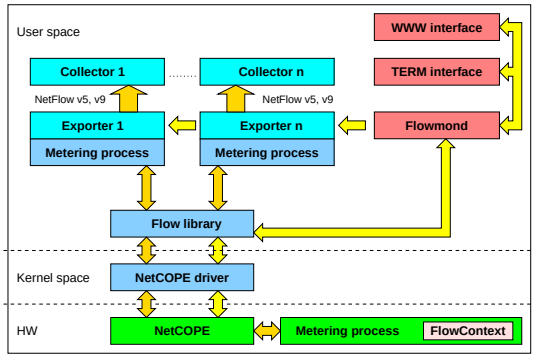


Figure 5.10: Layout of software with separate second metering processes

The kernel driver provides an abstract layer of the underlying PCI architecture by interface for read and write operations, booting the firmware and DMA transfers. The DMA transfers are triggered by the firmware via interrupt when enough flow-record is ready to be transferred or when the timer overflows.

The common flow library allows applications to access the flow-records after they are in the PC memory. Driver shares this memory with application so there is no data duplication which optimize the performance of applications. Further the library implements functions that are able to configure the probe parameters such as inactive and active timeout, samplings, etc.

Because of the collisions of flow-records in the memory of firmware there has to be a second stage aggregation process that allows to put together fragmented-flows. The process must aggregate all flows without collisions. To this end the indirect hashing with lookup is a good choice. Also the bidirectional ordering list shows up as an efficient software implementation of inactive timeout heuristic. The metering process is ideally suited right

into the common library where it can process all the firmware flow-records only once and then to distribute them to the applications. Unfortunately it would spoil the concept of shared memory between driver and applications, therefore other solutions might be also possible. For example to connect the aggregation process to each exporter (see Fig. 5.10).

The architecture of exporter must be modular to allow later improvements and additions. Individual modules can implement various processing of flow-record like:

- *Filtering* – is a basic interval matching related to network addresses in the record. If they are within the range then the flow-record is accepted.
- *Anonymization* – is designed to hide original data source information, like IP addresses and port numbers.
- *Protocol for Flow Export* – can be changed to NetFlow v5, v9 or IPFIX.

At the end flow data are sent using UDP, TCP or SCTP channel to the collector. The probe is remotely configured via the terminal or web interface using the NETCONF protocol [16].

5.5 Parameters Settings

One of the most difficult tasks is to configure the probe settings to suit the administrator needs and to allow maximum performance with current traffic distribution. The probe has several parameters which could be set on-the-fly. For example inactive and active timeouts, input sampling range and mode, output sampling of flow-records and various thresholds that triggers another functions. For example even the adaptive sampling must be initialized with sampling values that are used according to triggers.

Given that the incoming traffic is variable during time, the optimal settings is hard to find and usually it depends on the experience of the administrator. Therefore we suggest to either assist the administrator via advertising important statistics about current network traffic mix or to implement a configuration process that observes traffic for a short time-interval and then set the probe parameters automatically.

To this end the acceleration card must be able to switch its functionality between ordinary network interface card and flow monitoring probe. The configuration process can observe all the traffic coming on the network interface at first, then switch the card to the flow monitoring probe, configure it and observe if the probe follows the anticipated behavior. The flowchart on the Fig. 5.11 shows the whole configuration process.

5.6 Concept of Variable Flow-Record

Currently the flow monitoring probes have fixed structure of flow-record. This state corresponds with the old popularity of NetFlow v5 which has a fixed format of export protocol. On the other hand new flow export protocols are ready to transfer any information elements. To this end we propose the concept of variable flow-record. It requires that all components which works with the flow-record are aware of changes of the structure and are able to process the modified flow-record.

XML scheme is suggested to define the structure of the flow-record, unified header, operations and controls. The description of XML definition is given in Fig. 5.12. The XML tree shows how aggregation operations (`<flowoperation>`) are assigned to the field in the

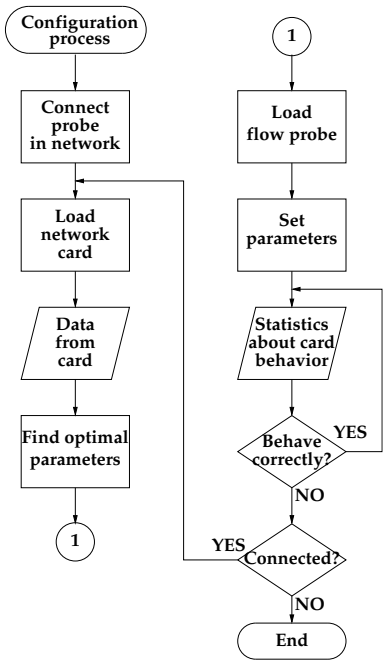


Figure 5.11: Flowchart of configuration process

flow record (`<flowfield>`). Each field is also paired with its modifier represented by field in the UH record (`<uhfield>`). Additionally a flowfield can be paired with control operation (`<flowcontrol>`) which can drive its expiration.

Complete DTD validation scheme and an example of XML configuration file for the whole flow-record are attached in Annex C. The XML configuration file must be passed to several units so they recognize and process the flow-record correctly.

The Fig. 5.13 shows which components needs to be aware of XML configuration. It is supposed that once the administrator configure his record, new firmware is generated and uploaded to the FPGA. It leads to significant savings of FPGA resources but also to the loss of current statistics if the monitoring is already running and the firmware must be reloaded. Following list describes what kind of information has to be passed to each component.

- *HFE-C* is generated according to the information about Unified Header. Particularly, the generator is interested in what fields must be extracted out of the packet and where is their location in Unified Header.
- *Hash Generator* must be aware of the key-fields in the Unified Header to be able to generate the Flow ID.
- *Flow Processing Unit* is the most complex. It utilizes the XML reference mechanism to associate the fields in flow-record with appropriate field in UH and other tags defining control and aggregation operations.
- *Second Aggregation process* must be aware of the flow record structure to process it and aggregate correctly.
- *Exporter* needs to know the structure of the flow-record to use it when the exported packets are filled with data.

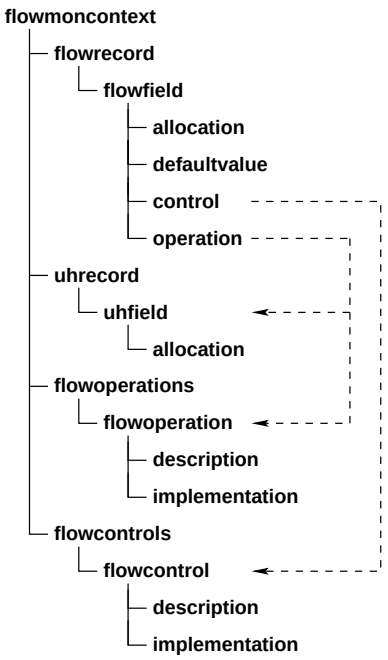


Figure 5.12: XML tree with cross references

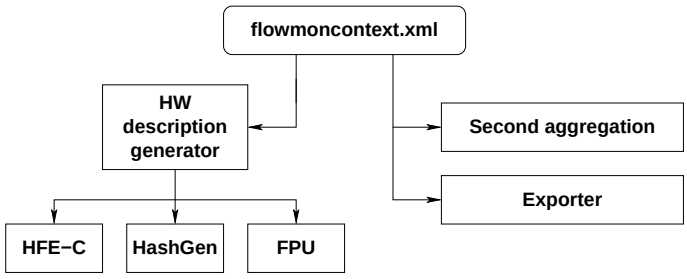


Figure 5.13: XML usage by system components

Chapter 6

Models

The architecture of the probe must be robust and comply with required performance. Robustness of the probe is its ability to stay operational even during the attacks against the probe resources. Sufficient performance must allow to process a typical ten gigabit traffic without a packet loss. It is usually hard to prove such characteristics in an exact way. But modeling and simulations can reveal some of the bottlenecks which limit the performance. It is not feasible to build whole model of the probe with all the details as it would require higher effort then the system implementation itself. Therefore the architecture is decomposed into smaller parts where irrelevant details are omitted. It is worth to model only those parts that can influence the performance or the behavior of the probe. Also some of the parts do not have to be modeled since their behavior can be inferred easily from known information.

6.1 Packet Parsing

The packet parsing is performed by eight HFE-C units as described in Sec. 5.3.2. The sequence splitter distributes packets to small buffers before each HFE-C. The policy of packet assignment to HFE-C is simple: Transfer the packet to the least occupied buffer. Each HFE-C generates Unified Header from packet to its dedicated buffer in sequence binder where it waits till it is read out in correct order.

The least occupied buffer is the best policy to distribute packets among HFE-C units because it causes the smallest ordering errors. The main reason for this statement is that if the variable length of packets is omitted then "the least occupied buffer" policy equals to the round-robin fair distribution all HFE-C has same performance (one word of packet per cycle). The variable length of the packet is taken into account by introducing an additional criterion of buffer utilization. As a consequence it allows to keep the input buffers of sequence binder reasonably small.

It is suggested to size the input buffers of HFE-C to host the longest packet. So the splitting process does not have to wait till the longest packet is processed by HFE-C. The setup is analyzed on three types of network traffic that can possibly emerge:

- Only the longest packets – Packets are sequentially assigned to HFE-C units. When the first assigned HFE-C is again about to be assigned, its input buffer is already empty.
- Only the shortest packets – Same as previous point. Buffers are less utilized.

- Mix of short and long packets – HFE-C are able to process all of them, but ordering errors happen.

The worst case scenario happens when the longest packet is outrun by the shortest ones. According to this scenario each buffer of Sequence Binder must be sized to host at least $|LongestPacket|/|ShortestPacket|$ of Unified Headers ("$|packet|$" means the length of the packet). The example is given for case of one long and several short packets on Fig. 6.1.

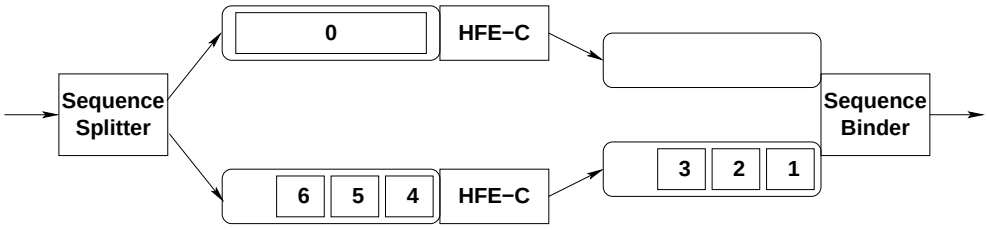


Figure 6.1: Headers with sequence numbers 1, 2, 3 wait for header of long packet 0

Neither splitter nor HFE-C are bottlenecks in the throughput for ten gigabit traffic. The Sequence binder could cause congestion of its buffers because it has to wait for headers of long packets to process them in correct order. Fortunately the size of headers is the same as of the shortest packets. Therefore if the buffers of binder are full with headers of short packets (blocked by the header of a long one) then another subsequent long packet causes a time window for emptying of these buffers.

6.2 Temporal locality and Memory Access

The FlowContext unit handles the storage and distribution of context information. It can be connected to any type of memory available on COMBO6X card. It is either slow but large DRAM or smaller and faster SSRAM or on-chip very small and the fastest BlockRAM (BRAM). Unfortunately smaller memories cannot be used as a main storage for all simultaneous flow-records and therefore only DRAM can be used. A more sophisticated design should use a hierarchical system of memories in a similar way as the processor cache is used by the PC architecture. To this end, we follow the presumption stated in Sec. 4.2.5 and explore the temporal locality of the traffic from the flow point of view. A good temporal locality would support the implementation of cache and DRAM. The parameter of interest is the flow gap (see Fig. 6.2), i.e., the number of foreign packets between two packets of the same flow.

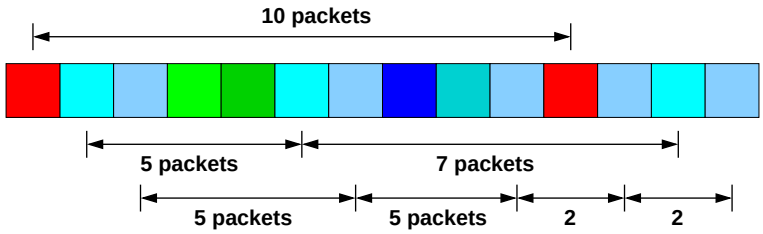


Figure 6.2: Examples of flow gaps

The measurement consists of collecting traffic samples from several points of CESNET [1] network and their analysis. Tcpdump tool was used to store packet traces on the

disk. Each trace contains source and destination IP address, source and destination port (if applicable) and protocol number, an example is given:

```
147.251.145.17:1390,130.14.29.110:80
147.32.122.243:,132.229.216.165:
```

Please note that the traces are without timestamps. Although tcpdump can assign timestamp to each packet, its resolution is limited to milliseconds which is not enough to distinguish between two but also between thousands of packets on ten gigabit network. Another reason not to use timestamps is that the point of interest are flow-gaps which are perfectly measurable with relative position in time, i. e. time measured in number of packets.

Following statistical indicators are measured for every flow:

- Number of packets
- Average flow gap
- Minimum and maximum flow gap
- Standard deviation of the flow gap
- Histogram of the flow gap

The overall statistics consist of weighted averages (the more packets, the greater weight) of maximum, minimum and average flow gaps. Cumulative histograms are constructed to display the portion of the traffic with the flow gap below a certain threshold. Fig. 6.3 shows an example of cumulative histogram of weighted maximum gaps.

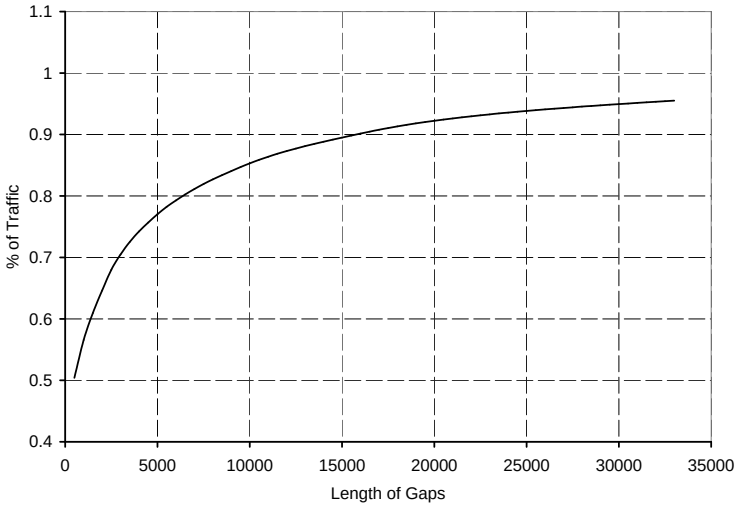


Figure 6.3: Distribution of maximum flow-gaps

Measurements showed that the flow gap is exponentially distributed. Which means that most of them are very short whereas long flow-gaps are rare. According to the graph we can set the size of the cache to the longest flow gap for which we still want to keep the flows in the cache. And vice versa, the cumulative histogram also allows to estimate percentage of the traffic that requests its flow-records from the cache.

Table 6.1: Number of cycles to retrieve 64 B flow-record from various memories

Memory	**Access time**	**Read time**	**Total**
Internal Cache (BRAM)	1	8	9
External Cache (SSRAM)	4	8	12
No cache (only DRAM)	16	8	24

COMBO6X card allows to implement cache either in external SSRAM or on-chip small BlockRAMs. The access times including time to read out the whole record are given in Tab. 6.2.

According to the Amdahl's law, the speed-up factor of whole process is

$$A = \frac{1}{(1-f) + \frac{f}{r}} \tag{6.1}$$

where f is the part of task that is accelerated and r is the speed-up of the part.

Graph on Fig. 6.4 shows the speed-up for increasing size of the cache. Please note that cache with up to 4096 of flow-records can be implemented in FPGA and larger one in external SSRAM memories.

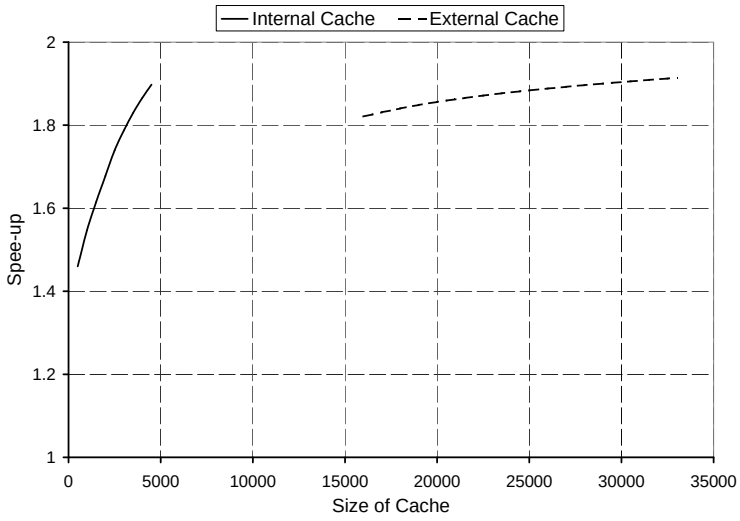


Figure 6.4: Estimated speed-up of Cache with DRAM

Apart from estimating the cache size, we also want to tune its performance. The simulations are focused on the on-chip cache since no optimizations can be implemented using an external cache. Caches have several parameters such as associativity, victim politics and others. The degree of associativity is limited by available implementations in FPGAs. Though we have simulated several degrees of associativity with the fixed size of the cache to determine whether the higher degree of associativity leads to significant improvements. The graph on Fig. 6.5 shows that it is not worth to implement high degree of associativity as it does not significantly improve the performance of the cache. At the same time LRU victim policy was compared against the random one. Again LRU does not perform significantly better than random victim policy (only difference of three percents).

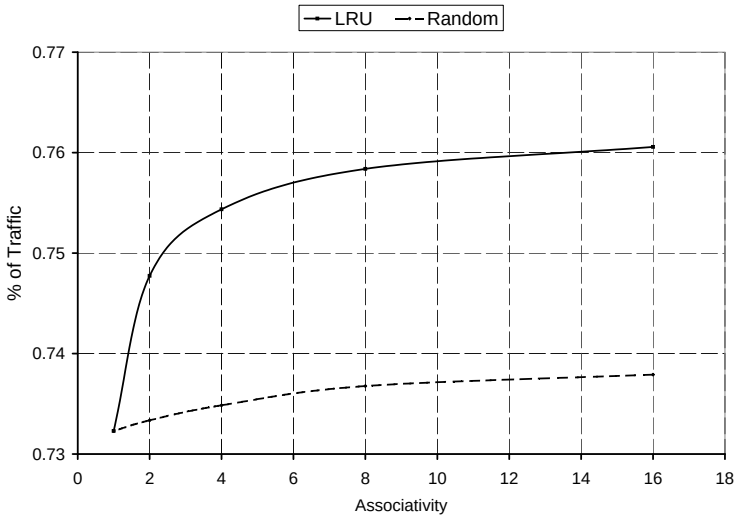


Figure 6.5: Simulation results of associativity and victim policy

In the previous paragraphs it was find out that the maximum speed-up to retrieve the flow-record is around 1.8 of the original time when using the on-chip cache. The original time is 24 cycles whereas the new one is 15 cycles. Though it is not enough to process full ten gigabit traffic consisting of the shortest packets only. It would require only 4 cycles. Fortunately a typical ten gigabit traffic is far less packet-intensive as it consists of the mix of long and short packets. Therefore it is a decision to make whether use a cache solution or not as the implementation costs could be high. Following Tab. 6.2 gives an estimated throughput of both solutions for increasing lengths of packets.

Table 6.2: Estimated throughput[Gbps] of FlowContext

Length	64	96	128	192	256
Throughput(cache)	4.4	6.6	8.8	10.0	10.0
Throughput(DRAM)	2.7	4.1	5.4	8.2	10.0

6.3 Collisions, fragmented flows and deceleration

The concept of the whole probe is based on the processing of the incoming traffic in the firmware where it is decelerated so the outgoing data stream can be handled in software. Available memory capacity for flow-records limits the aggregation on the card. But it is not the only one. Suggested direct addressing of flow-records by hash value is another parameter that limits the aggregation factor. In fact the memory capacity is sufficient to hold all simultaneous flow-records if there was a perfect hash that could distinguish all flows with limited bit range or different type of addressing. Of course such hash function cannot exist because if a fully occupied larger state space is transformed by hash into smaller one then collisions must happen. The probability of collision for direct addressing can be expressed as:

$$P(collision) = \frac{N}{C} \tag{6.2}$$

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

A typical ten gigabit traffic consists of 100'000 flows approximately. Therefore the probability of collisions is very high for small memories. And despite that even small memory can provide certain level of aggregation. It is due to the burstiness and fact that 10%–20% of flows account for 90% of total traffic. Thus the deceleration factor is influenced by the mix of:

- collisions of heavy flows,
- collisions of light and heavy flows and vice versa,
- collisions of light flows.

Because it is hard to estimate the weights of each case in the deceleration function and also the exact distribution of flow types, it is better to simulate the behavior. The deceleration factor is derived from simulations as

$$Deceleration = \frac{TotalPackets}{collisions} \tag{6.3}$$

Fig. 6.6 displays deceleration function for small sizes of memory. The curve shows how small memories suffer from a lot of collisions of heavy flows which becomes better with increasing sizes until the influence disappears completely.

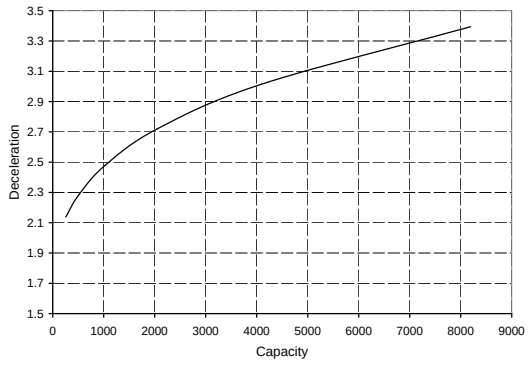


Figure 6.6: Deceleration factor for small sizes of memories

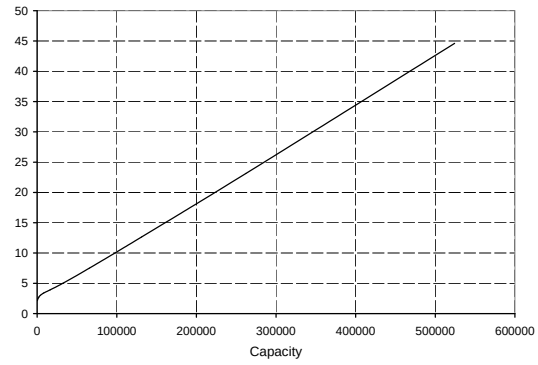


Figure 6.7: Deceleration factor for large sizes of memories

The second graph on Fig. 6.7 represents the deceleration function for larger memories. The deceleration is linearly dependent on the size of larger memories because of negligible influence of heavy collisions which are rare. The deceleration function can be expressed as

$$Deceleration = c * \frac{1}{P(collision)} \tag{6.4}$$

where c is a coefficient dependent on the incoming traffic mix.

Another graph on Fig. 6.8 shows how many flows is created when using different sizes of memories. Again small memories create a lot of flows (so called fragmented-flows) because of high number of collisions. As the size of memory increases and the number of collisions decreases, the number of flows settles on a nearly fixed value. The situation is displayed 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

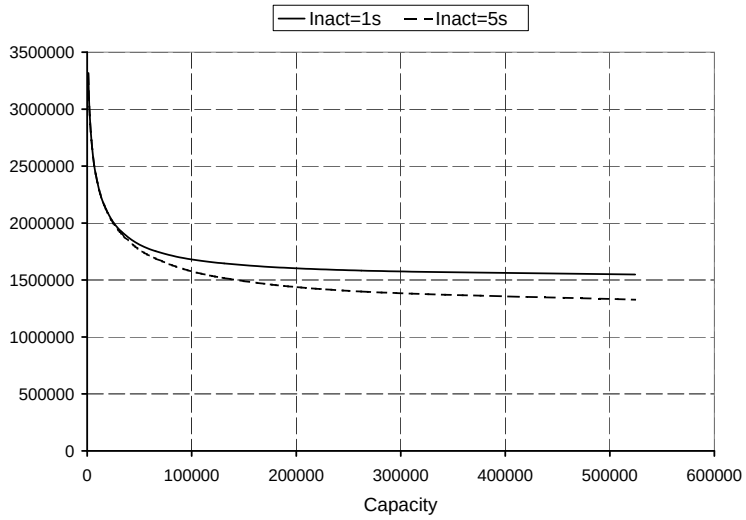


Figure 6.8: Number of created flows depending on the size of memory for representative sample of the traffic

by shorter timeout marked as inactive. Despite that the number of created flows remains steady around a fixed value. It tells us that the number of real existing flows in the traffic is reached and most likely a lot of flows is not fragmented.

Obtained results indicate how the memories of COMBO6X card are to be utilized for the aggregation process. Memory sizes of COMBO6X are given in Tab. 6.3 (size is measured in number of 64 Bytes flow-records).

Table 6.3: Types of memories with their capacity

Memory type	**Number of records**	**Deceleration factor**
BRAM(FPGA)	4096	~ 2
3xSSRAM	96 K	~ 10
DRAM	512 K – 8 M	~ 40 – ~ 200

There are two possibilities where to implement main memory of flow-records. It is either SSRAMs or DRAM, BlockRAM memory is too small to give reasonable deceleration factor. The SSRAM capacity can decelerate the typical ten gigabit traffic ten times. It might seem to be enough but if a fully loaded link is considered then the bulk of transfered data to PC is too high (more than 200Mbps). Therefore the only solution is to utilize the DRAM with high capacity. The design of memory utilization is following:

- *DRAM* – main memory for flow-records
- *BRAM* – cache to support main memory
- *SSRAMs* – memory of flow states, utilized by Flow State Manager

Such a lay-out has a high deceleration factor (about 250) as well as a good throughput (see Tab. 6.2). SSRAMs are utilized by the Flow State Manager for keeping state of flows using the timestamp algorithm.

6.4 Resource Protection using Sampling

The probe is expected to be reliable and robust. It means not only long times between errors but also that the probe must be able to monitor unexpected traffic mix even though it is not designed to do so from the long-term point of view. Malicious traffic occurs during attacks (DoS, port scans, smurfs and others) or network anomalies (such as routing failures).

Despite the probe is able to monitor part of the malicious traffic even without any protective method, most of the packets are nondeterministically discarded which makes it impossible to estimate the original traffic mix.

To this end the sampling algorithm is proposed to protect the probe hardware and software resources. It means that the sampling must be able to decrease the incoming data rate and also to decrease the memory requirements. At the same time it must not introduce large bias in measured statistics. Therefore it is supposed that the current sampling rate is advertised to all analyzing programs running at the collector. The original traffic mix can be mostly extrapolated from sampled statistics. For example what concerns of number of packets or bytes it is sufficient to multiply measured results by reciprocal value of sampling rate. On the other hand it is much more harder to estimate the original number of flows. It is because the sampling change distribution of packets in flows and so a lot of the light flows disappear at all, while the heavy flows remains all. This behavior has a positive effect on the occupation of the flow-record memory as required. On the other hand it can make certain analysis technique not applicable due to the bias in number of flows statistics. Fig. 6.9 shows an example how the sampling influences the distribution of packets in flows.

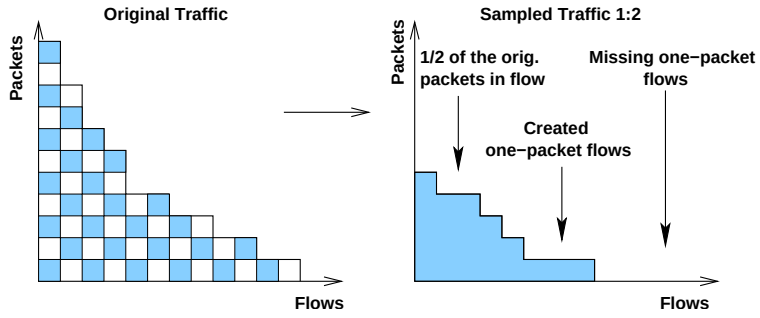


Figure 6.9: Influence of sampling on distribution of packets in flows

As an example let us consider how to set the sampling rate for the estimated throughput of FlowContext unit with cache (see Tab. 6.2). The unit is able to process maximum of 7 million of packets per second which is 4.4 Gbps using the shortest packets only. Thus if the sampling rate is set to the rate of 1/3 then even ten gigabit traffic with 15 million of shortest packets cannot overwhelm the unit. The FlowContext unit stores flow-records in the memory which has limited capacity. Let us suppose that it is designed to handle 100'000 simultaneous flows with inactive timeout 10 second. It means that the traffic can create at most 10'000 new flows every second while the other 10'000 must expire. The administrator expects the malicious traffic to create 1'000'000 one-packet flows per second and therefore he sets the sampling rate to 1/100. Such a high static sampling rate is an issue because the probe is supposed to work with the typical traffic mix most of its time and it can be monitored without any sampling.

The adaptive sampling is proposed to suppress disadvantages of static sampling. It is supposed that the sampling rate is driven by current packet rate and memory utilization to

meet the demands on limited throughput and memory capacity. A priority is always given to request for decrease of the sampling rate. The implementation is done using a range associative array of parameters and sampling rates. The concept is described in Sec. 5.3.1. It is proper to set sampling rates according to the exponential function of low order. For example 2^x. It allows to predict very easily what happens after the range of current row is exceeded. As an example, let us again consider the case of the FlowContext unit. The upper ranges of packet rate are set to limit the speed of the traffic, so it stays within the throughput of the unit. It means that if the sampling is applied the resulting packet rate must be below 7 million packets per second. The memory utilization must not exceed memory capacity and therefore it is set similarly to packet rate with small modification. Because the current memory utilization is measured using already sampled traffic, the limits of ranges must respect the results caused by decreased/increased sampling. A small example of four configuration rows is given in Tab. 6.4. To explain how the proposed

Table 6.4: Example of configuration of Adaptive unit

Packet Rate		**Mem. Utilization**		**Sampling**
Lower	Upper	Lower	Upper	
0 mil.	7 mil.	0%	60%	1
5 mil.	14 mil.	20%	70%	2
13 mil.	15 mil.	25%	80%	4
13 mil.	15 mil.	30%	85%	8

configuration works let us imagine following scenario. A typical traffic is being monitored with sampling rate equal to one. Suddenly a DoS attack (one-packet flows) emerges. For simplicity the packet rate remains the same but the memory utilization exceeds 60% of memory capacity. Therefore the sampling is decreased to 1:2. Please note that there is a small margin between upper limits of adjacent rows which allows the probe to expire old flows while new are already monitored without further decreasing of the sampling rate. The minimal expected utilization after decreased sampling rate is 30 percent which is above the lower limit so the sampling rate is not immediately increased back to one. Now if the DoS continues, the memory keeps filling again and the sampling rate is decreased.

Of course other schemes of configuration can be used and may perform better (less biased statistics) as the limits do not have to be so strict if the malicious traffic is expected to be less aggressive.

Chapter 7

Conclusion

This thesis presented an IP-flow monitoring probe intended for 10-Gigabit networks. The design of the probe was preceded by the study of flow based monitoring. The stress was given on algorithms and heuristics whose good understanding allows to optimize and distribute monitoring tasks between target platforms efficiently. The probe was designed for COMBO acceleration cards and PC. Such a combination allows to accelerate time-critical part on the card and additional operations can be done in PC.

The contribution of proposed probe is its ability to either monitor whole bandwidth or to discard deterministically the out-of-band traffic and to inform about it subsequent systems. It is significant step forward in comparison to other implementations. Next contribution is design and implementation of adaptive sampling algorithm with suggestions on its configuration. Further, a study was given on the temporal locality of the ten gigabit traffic with suggestions on how to use it for memory hierarchy of the probe. The thesis also gives a basis to the definition of variable structure of the flow-record which allows administrator to monitor various additional characteristics of the traffic. Therefore the implementation of control mechanisms must be generated on demand using Handel-C.

The system architecture was optimized to suit the COMBO cards hardware, especially the memory architecture, so to allow maximum performance. Simulations of critical parts of the architecture allowed to estimate the throughput of the architecture. It is more than sufficient to monitor typical 10-Gigabit network link.

Target applications of the probe are in the field of security where a reliable measurement is vital for analysis systems. For example, flow data can be used to build users profiles and then to detect anomalies of the original behavior. It is also possible to look for DoS and other flooding attacks with various methods (for example [20], [7]).

The future work is focused on further optimizations of the architecture to increase its performance and scalability. Our interest is also in sampling algorithms where methods of estimating the original traffic mix should be further investigated. A potential outcome of the study might be an algorithm to estimate not only the original number of packets or bytes but also the original number of flows with their traffic distribution. Next the exact mathematical model of adaptive sampling algorithm should be created so the configuration parameters could be derived.

Bibliography

[1] Cesnet web pages, http://www.cesnet.cz, 2007.

[2] Cisco web pages, http://www.cisco.com, 2007.

[3] Liberouter web pages, http://www.liberouter.org, 2007.

[4] L. Deri. nprobe an extensible netflow v5/v9/ipfix gpl probe for ipv4/v6. http://www.ntop.org/nprobe.html, 2006.

[5] B. Aboba, J. Arkko, and D. Harrington. Introduction to accounting management. RFC 2975, Internet Engineering Task Force, October 2000.

[6] D. Awduche, J. Malcolm, J. Agogbua, M. O'Dell, and J. McManus. Requirements for traffic engineering over MPLS. RFC 2702, Internet Engineering Task Force, September 1999.

[7] P. Barford, J. Kline, D. Plonka, and A. Ron. A signal analysis of network traffic anomalies. In *IMW '02: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pages 71–82, New York, NY, USA, 2002. ACM Press.

[8] K. C. Claffy, H. Braun, and G C. Polyzos. A parameterizable methodology for internet traffic flow profiling. *IEEE Journal on Selected Areas in Communications*, 13(8):1481–1494, 1995.

[9] B. Claise. Cisco systems netflow services export version 9. RFC(Informational) 3954, Internet Engineering Task Force, 2004.

[10] B. Claise. Ipfix protocol specification. Internet draft: draft-ietf-ipfix-protocol-21.txt, work in progress, Internet Engineering Task Force, April 2006.

[11] S. Deering and R. Hinden. Internet protocol, version 6 (ipv6) specification. RFC 2460, Internet Engineering Task Force, December 1998.

[12] T. Dietz, F. Dressler, G. Carle, and B. Claise. Information model for packet sampling exports. Internet draft: draft-ietf-ipfix-info-07, work in progress, Internet Engineering Task Force, May 2004.

[13] N. G. Duffield and M. Grossglauser. Trajectory sampling for direct traffic observation. *IEEE/ACM Trans. Netw.*, 9(3):280–292, 2001.

[14] N. Duffield. A framework for packet selection and reporting. Internet draft, Internet Engineering Task Force, 2005.

[15] N. Duffield and C. Lund. Predicting resource usage and estimation accuracy in an ip flow measurement collection infrastructure. In *IMC '03: Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, pages 179–191. ACM Press, 2003.

[16] R. Enns. Netconf configuration protocol. RFC 4741, Internet Engineering Task Force, 2006.

[17] C. Estan, K. Keys, D. Moore, and G. Varghese. Building a better netflow. *SIGCOMM Comput. Commun. Rev.*, 34(4):245–256, 2004.

[18] C. Estan and G. Varghese. New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. *ACM Trans. Comput. Syst.*, 21(3):270–313, 2003.

[19] J. Fan, J. Xu, M. H. Ammar, and S. B. Moon. Prefix-preserving ip address anonymization: measurement-based security evaluation and a new cryptography-based scheme. *Comput. Networks*, 46(2):253–272, 2004.

[20] Y. Gu, A. Mccallum, and D. Towsley. Detecting anomalies in network traffic using maximum entropy estimation. Internet Measurment Conference, p. 345–350, 2005.

[21] M. Košek. Stavové zpracování tcp/ip toků. Bachelor thesis, FIT BUT, 2007.

[22] G. Minshall. Tcpdpriv command manual, 1996.

[23] M. Molina, A. Chiosi, S. D'Antonio, and G. Ventre. Design principles and algorithms for effective high speed ip flow monitoring. Technical report, 2004.

[24] J. Postel. User datagram protocol. RFC 768, Internet Engineering Task Force, August 1980.

[25] J. Postel. Internet protocol. RFC 791, Internet Engineering Task Force, September 1981.

[26] J. Postel. Transmission control protocol. RFC 793, Internet Engineering Task Force, September 1981.

[27] J. Quittek, T. Zseby, B. Claise, and S. Zander. Requirements for IP flow information export (IPFIX). RFC 3917, Internet Engineering Task Force, October 2004.

[28] G. Sadasivan, N. Brownlee, B. Claise, and J. Quittek. Architecture for ip flow information export. Internet draft: draft-ietf-ipfix-architecture-08.txt, work in progress, Internet Engineering Task Force, March 2005.

[29] J. Tobola. Platforma pro rychlý vývoj síťových zařízení. Master thesis, FIT BUT, 2007.

[30] T. Košnár. Notes to flow-based traffic analysis system design. Technical report, CESNET, 2004.

[31] T. Zseby, E. Boschi, N. Brownlee, and B. Claise. Ipfix applicability. Internet draft: draft-ietf-ipfix-as-10.txt, work in progress, Internet Engineering Task Force, August 2006.

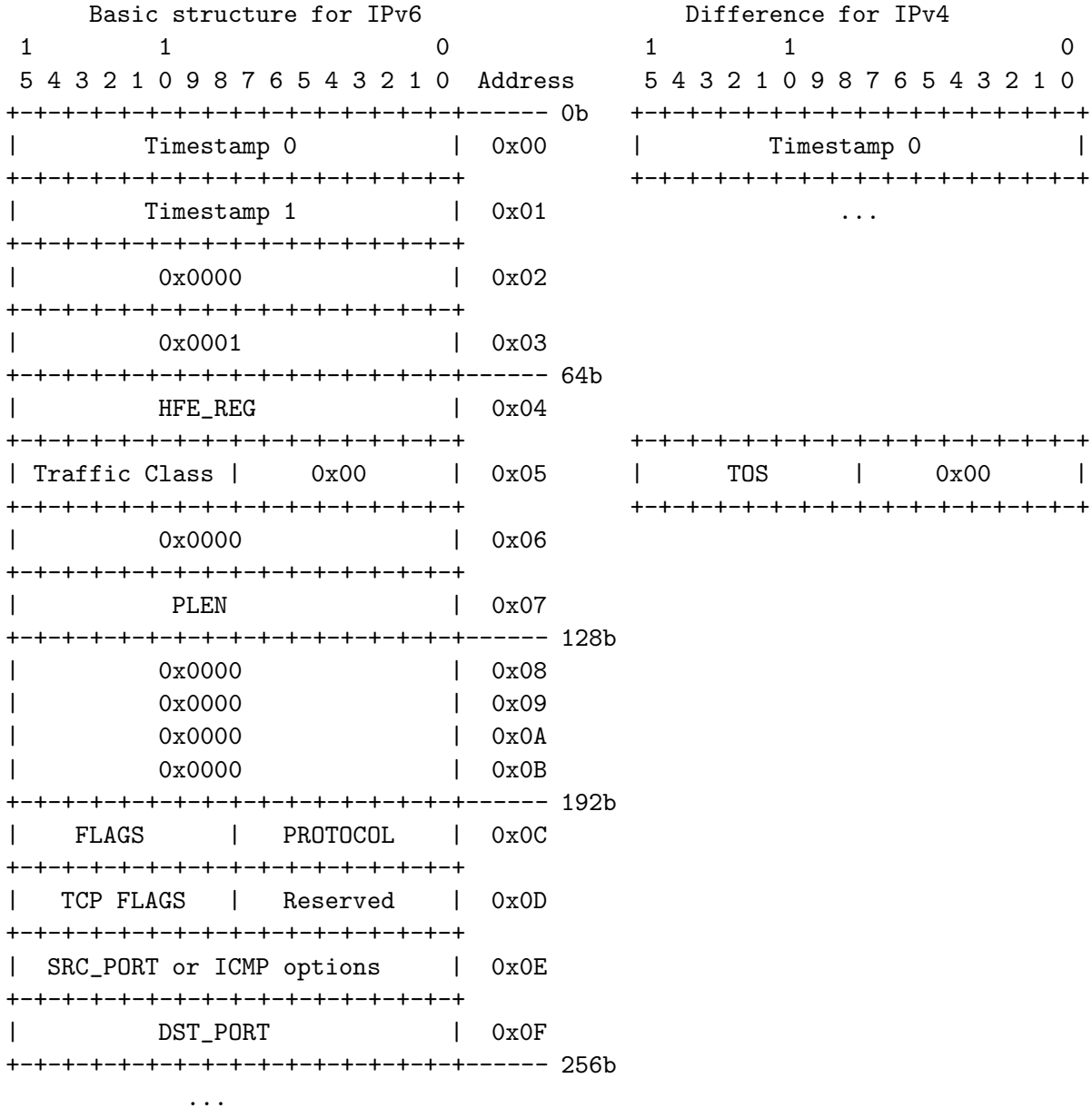
Annex A

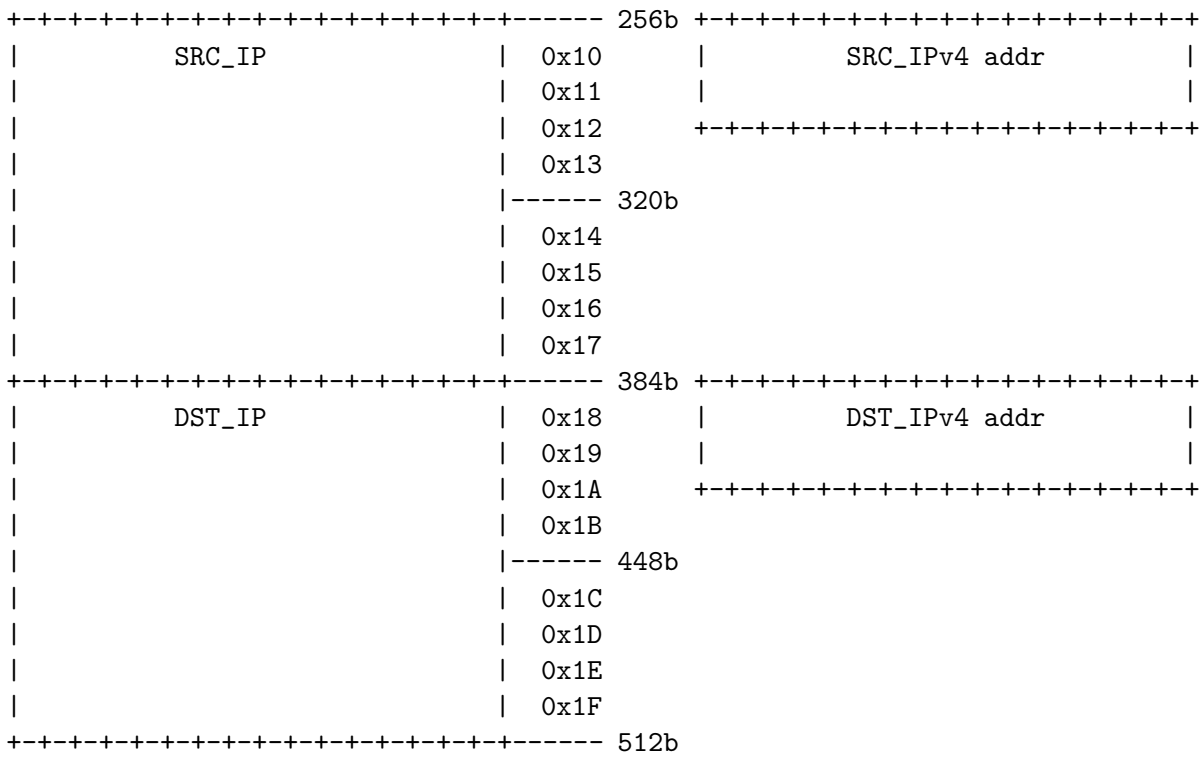
Table 7.1: Version 5 Flow Record Format

Flow-key	Type	Description
yes	srcaddr	Source IP address
yes	dstaddr	Destination IP address
no	nexthop	IP address of next hop router
no	input	SNMP index of input interface
no	output	SNMP index of output interface
no	dPkts	Packets in the flow
no	dOctets	Total number of Layer 3 bytes in the packets of the flow
no	First	SysUptime at start of flow
no	Last	SysUptime at the time the last packet of the flow was received
yes	srcport	TCP/UDP source port number or equivalent
yes	dstport	TCP/UDP destination port number or equivalent
no	tcp_flags	Cumulative OR of TCP flags
yes	prot	IP protocol type (for example, TCP = 6; UDP = 17)
no	tos	IP type of service (ToS)
no	src_as	Autonomous system number of the source, either origin or peer
no	dst_as	Autonomous system number of the destination, either origin or peer
no	src_mask	Source address prefix mask bits
no	dst_mask	Destination address prefix mask bits

Annex B

Example of Unified Header.




256b
SRC_IP
0x10
0x11
0x12
0x13
320b
0x14
0x15
0x16
0x17
384b
SRC_IPv4 addr
DST_IP
0x18
0x19
0x1A
0x1B
448b
0x1C
0x1D
0x1E
0x1F
512b
DST_IPv4 addr

Annex C

The DTD scheme of the XML configuration of the variable flow-record.

```
<!ELEMENT flowmoncontext (flowoperations,flowcontrols,uhrecord,flowrecord)>

  <!ELEMENT flowoperations (flowoperation+)>
  <!ELEMENT flowcontrols   (flowcontrol+)>
  <!ELEMENT uhrecord   (uhfield+)>
  <!ELEMENT flowrecord (flowfield+)>

  <!ELEMENT flowoperation  (description,implementation?)>
  <!ELEMENT flowcontrol    (description,implementation?)>
  <!ELEMENT uhfield    (allocation,defaultvalue?)>
  <!ELEMENT flowfield  (allocation,defaultvalue,operation,control*)>

  <!ELEMENT description    (#PCDATA)>
  <!ELEMENT implementation (#PCDATA)>

  <!ELEMENT allocation    (EMPTY)>
  <!ELEMENT defaultvalue  (EMPTY)>
  <!ELEMENT operation (EMPTY)>
  <!ELEMENT control   (EMPTY)>

  <!ATTLIST operation
                   name       ID       #REQUIRED>

  <!ATTLIST control
                   name       ID       #REQUIRED>

  <!ATTLIST uhfield
                   name       ID       #REQUIRED
                   identifier (yes|no) no>

  <!ATTLIST flowfield
                   name       ID       #REQUIRED
                   identifier (yes|no) no>

  <!ATTLIST allocation
                   address    CDATA   #REQUIRED
                   size       CDATA   #REQUIRED>
```

```
  <!ATTLIST defaultvalue
                    value       CDATA    #REQUIRED>

  <!ATTLIST operation
                    name        IDREF #REQUIRED
                    operand1    IDREF #IMPLIED
                    operand2    IDREF #IMPLIED
                    constant    CDATA #IMPLIED>

  <!ATTLIST control
                    name        IDREF #REQUIRED
                    operand1    IDREF #IMPLIED
                    operand2    IDREF #IMPLIED
                    constant    CDATA #IMPLIED>
```

The XML configuration of the variable flow-record.

```
<?xml version=‘‘1.0’’ encoding=‘‘iso-8859-2’’ ?>
<flowmoncontext>
   <flowoperations>
      <flowoperation name=‘‘move-uh’’>
         <description>
            Always move value from unified header to the context.
         </description>
      </flowoperation>
      <flowoperation name=‘‘movefirst-uh’’>
         <description>
            Move value from unified header to the context only for the first
            packet of the flow.
         </description>
      </flowoperation>
      <flowoperation name=‘‘movelt-uh’’>
         <description>
            Conditional move. Move the value from unified header
            only if it is less than context value.
         </description>
      </flowoperation>
      <flowoperation name=‘‘moveneq-uh’’>
         <description>
            Conditional move. Move the value from unified header
            only if it is not equal to the context value.
         </description>
      </flowoperation>
      <flowoperation name=‘‘or-uh’’>
         <description>
            OR value from unified header with value in context and
            store it in context.
```

```
      </description>
   </flowoperation>
   <flowoperation name=‘‘and-uh’’>
      <description>
         AND value from unified header with value in context and
         store it in context.
      </description>
   </flowoperation>
   <flowoperation name=‘‘acc-uh’’>
      <description>
         Accumulate value from unified header with value in context
         and store it in context.
      </description>
   </flowoperation>
   <flowoperation name=‘‘acc-constant’’>
      <description>
         Accumulate constant specified in ‘‘source operation tag’’
         with value in context and store it in context.
      </description>
   </flowoperation>
</flowoperations>

<flowcontrols>
   <flowcontrol name=‘‘control-equal’’>
      <description>
         If the header and context field are not equal then replace the old
         flowrecord with the new one. The old record is exported to the
         NetCOPE swobuf.
      </description>
   </flowcontrol>
   <flowcontrol name=‘‘control-overflow’’>
      <description>
         If the value in context field are greater than specified constant
         in ‘‘source control tag’’ then expire the flowrecord.
         The record is exported to the
         NetCOPE swobuf.
      </description>
   </flowcontrol>
   <flowcontrol name=‘‘control-activetimeout’’>
      <description>
         If the value in difference of the actual time and the
         flowStratTimestamp is greater than active timeout than
         expire the flowrecord. The record is exported to the
         NetCOPE swobuf.
      </description>
   </flowcontrol>
</flowcontrols>
```

```
<uhrecord>
   <uhfield name=‘‘uh-timestamp’’ identifier=‘‘no’’>
      <allocation address=‘‘0x0’’ size=‘‘4’’/>
   </uhfield>
   <uhfield name=‘‘uh-constant-one’’ identifier=‘‘no’’>
      <allocation address=‘‘0x6’’ size=‘‘2’’/>
   </uhfield>
   <uhfield name=‘‘uh-hfeReg’’ identifier=‘‘no’’>
      <allocation address=‘‘0x8’’ size=‘‘2’’/>
   </uhfield>
   <uhfield name=‘‘uh-ipClassOfService’’ identifier=‘‘no’’>
      <allocation address=‘‘0xA’’ size=‘‘1’’/>
   </uhfield>
   <uhfield name=‘‘uh-ipPacketLength’’ identifier=‘‘no’’>
      <allocation address=‘‘0xE’’ size=‘‘2’’/>
   </uhfield>
   <uhfield name=‘‘uh-protocolIdentifier’’ identifier=‘‘yes’’>
      <allocation address=‘‘0x19’’ size=‘‘1’’/>
   </uhfield>
   <uhfield name=‘‘uh-tcpControlBits’’ identifier=‘‘no’’>
      <allocation address=‘‘0x1A’’ size=‘‘1’’/>
   </uhfield>
   <uhfield name=‘‘uh-portId’’ identifier=‘‘yes’’>
      <allocation address=‘‘0x1B’’ size=‘‘1’’/>
      <defaultvalue value=‘‘0x00’’/>
   </uhfield>
   <uhfield name=‘‘uh-sourcePort’’ identifier=‘‘yes’’>
      <allocation address=‘‘0x1C’’ size=‘‘2’’/>
   </uhfield>
   <uhfield name=‘‘uh-destinationPort’’ identifier=‘‘yes’’>
      <allocation address=‘‘0x1E’’ size=‘‘2’’/>
   </uhfield>
   <uhfield name=‘‘uh-sourceIpv4Ipv6Address’’ identifier=‘‘yes’’>
      <allocation address=‘‘0x20’’ size=‘‘16’’/>
   </uhfield>
   <uhfield name=‘‘uh-destinationIpv4Ipv6Address’’ identifier=‘‘yes’’>
      <allocation address=‘‘0x30’’ size=‘‘16’’/>
   </uhfield>
</uhrecord>

<flowrecord>
   <flowfield name=‘‘flowStartMicroseconds’’ identifier=‘‘no’’>
      <allocation address=‘‘0x0’’ size=‘‘4’’/>
      <defaultvalue value=‘‘0x00000000’’/>
      <operation name=‘‘movefirst-uh’’ operand1=‘‘uh-timestamp’’/>
      <control name=‘‘control-active-timeout’’/>
   </flowfield>
   <flowfield name=‘‘packetTotalCount’’ identifier=‘‘no’’>
```

```
   <allocation address=‘‘0x4’’ size=‘‘4’’/>
   <defaultvalue value=‘‘0x00000000’’/>
   <operation name=‘‘acc-constant’’ constant=‘‘0x000000001’’/>
   <control name=‘‘control-overflow’’ constant=‘‘0xFFFFF000’’/>
</flowfield>
<flowfield name=‘‘hfereg’’ identifier=‘‘no’’>
   <allocation address=‘‘0x8’’ size=‘‘2’’/>
   <defaultvalue value=‘‘0x0000’’/>
   <operation name=‘‘or-uh’’ operand1=‘‘uh-hfereg’’ operand2=‘‘hfereg’’/>
</flowfield>
<flowfield name=‘‘ipClassOfService’’ identifier=‘‘no’’>
   <allocation address=‘‘0xA’’ size=‘‘1’’/>
   <defaultvalue value=‘‘0x00’’/>
   <operation name=‘‘or-uh’’ operand1=‘‘uh-ipClassOfService’’
                          operand2=‘‘ipClassOfService’’/>
</flowfield>
<flowfield name=‘‘octetTotalCount’’ identifier=‘‘no’’>
   <allocation address=‘‘0xB’’ size=‘‘5’’/>
   <defaultvalue value=‘‘0x000000000’’/>
   <operation name=‘‘acc-uh’’ operand=‘‘uh-ipPacketLength’’/>
   <control name=‘‘control-overflow’’ constant=‘‘0xFFFFFF000’’/>
</flowfield>
<flowfield name=‘‘flowEndMicroseconds’’ identifier=‘‘no’’>
   <allocation address=‘‘0x10’’ size=‘‘4’’/>
   <defaultvalue value=‘‘0x00000000’’/>
   <operation name=‘‘move-uh’’ operand=‘‘uh-timestamp’’/>
   <control name=‘‘control-time-sequence’’/>
</flowfield>
<flowfield name=‘‘protocolIdentifier’’ identifier=‘‘yes’’>
   <allocation address=‘‘0x19’’ size=‘‘1’’/>
   <operation name=‘‘move-uh’’ operand=‘‘uh-protocolIdentifier’’/>
   <control name=‘‘control-equal’’/>
</flowfield>
<flowfield name=‘‘tcpControlBits’’ identifier=‘‘no’’>
   <allocation address=‘‘0x1A’’ size=‘‘1’’/>
   <defaultvalue value=‘‘0x00’’/>
   <operation name=‘‘or-uh’’ operand=‘‘uh-tcpControlBits’’/>
</flowfield>
<flowfield name=‘‘portId’’ identifier=‘‘yes’’>
   <allocation address=‘‘0x1B’’ size=‘‘1’’/>
   <operation name=‘‘move-uh’’ operand=‘‘uh-portId’’/>
   <control name=‘‘control-equal’’/>
</flowfield>
<flowfield name=‘‘sourcePort’’ identifier=‘‘yes’’>
   <allocation address=‘‘0x1C’’ size=‘‘2’’/>
   <operation name=‘‘move-uh’’ operand=‘‘uh-sourcePort’’/>
   <control name=‘‘control-equal’’/>
</flowfield>
```

```
      <flowfield name=‘‘destinationPort’’ identifier=‘‘yes’’>
         <allocation address=‘‘0x1E’’ size=‘‘2’’/>
         <operation name=‘‘move-uh’’ operand=‘‘uh-destinationPort’’/>
         <control name=‘‘control-equal’’/>
      </flowfield>
      <flowfield name=‘‘sourceIpv4Ipv6Address’’ identifier=‘‘yes’’>
         <allocation address=‘‘0x20’’ size=‘‘16’’/>
         <operation name=‘‘move-uh’’ operand=‘‘uh-sourceIpv4Ipv6Address’’/>
         <control name=‘‘control-equal’’/>
      </flowfield>
      <flowfield name=‘‘destinationIpv4Ipv6Address’’ identifier=‘‘yes’’>
         <allocation address=‘‘0x30’’ size=‘‘16’’/>
         <operation name=‘‘move-uh’’ operand1=‘‘uh-destinationIpv4Ipv6Address’’/>
         <control name=‘‘control-equal’’/>
      </flowfield>
   </flowrecord>

</flowmoncontext>
```

Annex D

The results in Fig. 7.1 indicate that the hardware probe is able to process 1 Gbps traffic at line rate without any packet losses, regardless of the packet size. In comparison, the nProbe [4] software suffers from massive losses for packet sizes below 400 B.

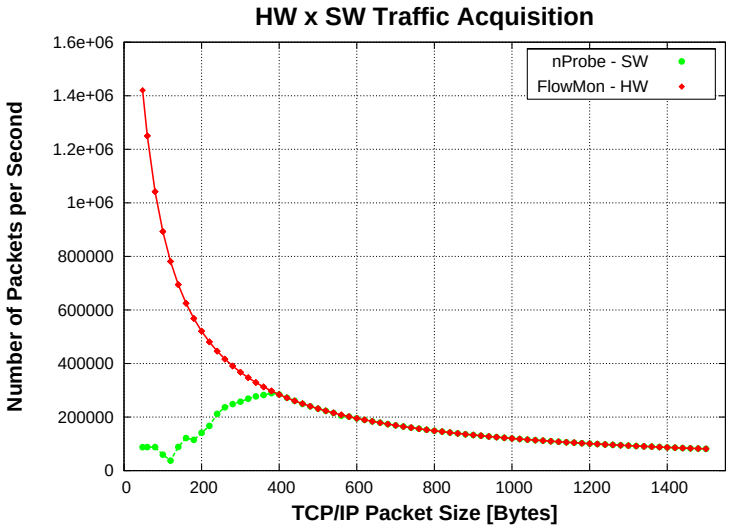


Figure 7.1: Software vs. hardware-accelerated 1 Gbps flow probes