Masaryk University

Faculty of Informatics

Next Generation Application-Aware Flow Monitoring

Doctoral Thesis

Petr Velan

Brno, Spring 2017

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Petr Velan

Advisor: doc. Ing. Pavel Čeleda, Ph.D.

Acknowledgement

This is the acknowledgement for my thesis, which can span multiple paragraphs.

Abstract

This is the abstract of my thesis, which can
span multiple paragraphs.

Main contributions:
- High-speed flow monitoring (100G, 2x80G)
- Addition of application layer information to flow monitoring
- Analysis of impact of traffic composition on application flow monitoring
- Proposal of new generation of application flow measurement - EventFlow

Other contributions
- Design of flexible flow collector for application flow processing
- Implementation of flexible IPFIX export for flowmon exporter
- Survey of methods for encrypted traffic classification

TODO: discuss privacy issues (where?)

Keywords

network, monitoring, network flow, NetFlow, IPFIX

Contents

1 Introduction

Motivation for flow monitoring: Rick's thesis (data retention!)
Use introduction from Challenges of Application Flow Monitoring

Rewrite, the following is copied from COMSOC article

A passive network traffic monitoring is an essential tool for network management, analysis, and security. There are several approaches to the passive network traffic monitoring which differ in the amount of reported information. On one hand, router interface counters can be reported using Simple Network Management Protocol to provide basic accounting information. On the other hand, deep packet inspection (DPI) provides detailed information about the content of the traffic and can be used for enforcing security policies or malware detection. The more detailed the analysis is, the more computing power it requires. Flow monitoring is a way of balancing the level of acquired information and necessary resources. Origins of flow monitoring date back to 1991 and the basic idea is to describe each network connection by a single record using the common properties of all packets in such a connection. Detailed information on the history and the current state of flow monitoring can be found in [1]. Due to the level of information aggregation, the flow monitoring can achieve 100 Gb/s throughput, and is, therefore, suitable for monitoring of large networks of internet service providers (ISP) and data centers.

Flow monitoring support is implemented in practically all enterprise routers and switches. It provides enough information to detect malicious or anomalous behavior and serves as a data source for many network security appliances. However, application-level attacks do not necessarily exhibit anomalous behavior on the network level and may remain undiscovered. As a DPI can discover these attacks more easily, flow monitoring is being enhanced with aspects of the DPI. The resulting method is called application flow monitoring (or measurement). Application flow records contain not only network level data but also information extracted from a payload of packets. Application flow monitoring can be seen as a compromise between the standard flow monitoring and full deep packet inspection: it balances performance and traffic visibility to provide as much information about the traffic as possible while being able to work on high-speed networks. The works of Cejka et al. and Husák et al. [2, 3] are examples of how application flow monitoring can be utilized to discover attacks on application protocol layer.

1.1 Network Monitoring

Little introduction to the network monitoring and especially the flow monitoring.
Introduce application flow monitoring as well, so that the reader knows what the RQs are about.

Anomaly detection and mitigation at internet scale: a survey
`http://dl.acm.org/citation.cfm?id=2525023.2525033`
`http://link.springer.com/chapter/10.1007/978-3-642-38998-6_7`
Popisuje kdo všecko umí měřit flow, ipfix, ...

1.2 Research Questions & Approach

1. How can flow measurement benefit from application layer information. [chap 3]
2. What is the performance impact of adding application layer information to flow monitoring. [chap 4]
3. How to monitor high-speed networks (100G+), what can be done to accelerate (application) flow monitoring. [chap 4]
4. Can application layer information be used to derive more information about the traffic (EventFlow). [chap 6]
5. How can flow monitoring cope with increasing share of encrypted traffic. [chap 5]

1.3 Contributions

1.4 Thesis Structure

Network Flow Monitoring - Introduction, explanation of the used terms and concepts. Serves as a thesis background.

2 Network Flow Monitoring

The article of Hofstede et al. [1] extensively covers the topic of flow monitoring process and much information provided in this chapter is influenced by it. The reader is encouraged to study the article, the relevant sections are II to VIII with the exception of section VII. However, this chapter explains the flow monitoring process in the context of this thesis, therefore some aspects of the flow monitoring, such as flow export processes, are described in more detail.

2.1 Flow Monitoring Basics

This section describes history and current state of network flow monitoring. We discuss related standards, introduce terminology used throughout this thesis and provide formal definition of a flow.

2.1.1 History of Flow Monitoring

The first mention of flow export can be found in RFC 1272 [4] published in 1991 by IETF Internet Accounting (IA) Working Group (WG). The goal of the document was to provide background information on Internet accounting. The authors describe methods of metering and reporting network utilization. The RFC defines a metering process as follows:

> *A METER is a process which examines a stream of packets on a communications medium or between a pair of media. The meter records aggregate counts of packets belonging to FLOWs between communicating entities (hosts/processes or aggregations of communicating hosts (domains)).*
>
> *Internet accounting: Background [4]*

The goal at the time was to provide a framework for traffic accounting, however, the common believe at the time was that internet should be free and any form of traffic capture, even for the accounting purposes, is undesirable. This, together with the lack of vendor interest, resulted in the conclusion of the working group in 1993. Note that the negative attitude towards the monitoring returns more than 20 years later [5].

In 1995, Claffy et al. showed a methodology for internet traffic flow profiling based on packet aggregation [6], which started a revival of flow monitoring efforts. The Realtime Traffic Flow Measurement (RTFM) Working Group was created in 1996 and it had three main objectives. First was to consider current issues relating to traffic measurement, such as security, privacy, policies and requirements on new network protocols. Second was to produce an improved Traffic Flow Model that should provide a wider range of measurable quantities (e.g. IPv6), simpler way to specify flows of interest, better access control to measured flow data, strong focus on data reduction capabilities and efficient hardware implementation. The third objective was to develop RTFM Architecture and Meter Management Information Base (MIB) as a standards track IETF documents. The effort resulted in 1999 by publishing several RFCs describing new traffic flow measurement framework with an increased flexibility and even a bi-directional flow support [7]. Since these documents fulfilled the objectives of the RTFM WG, the group was concluded in 2000. However, no flow export standard was developed as the vendors showed no interest in this area.

Meanwhile, Cisco realized that similar kind of flow information is already stored in a flow cache of their packet switching devices. The purpose of this cache is to speed up packet switching by making a forwarding decision only for the first packet of each flow. Unlike the RTFM flow measurement framework, the primary purpose of flow cache is not accounting nor monitoring, therefore the configuration of measurement process using a flow cache in a switch is severely limited. Despite the limitations, once Cisco introduced its own flow export technology called NetFlow, it achieved widespread adoption. The main reason for the extensive adoption was the fact that it was readily available in most Cisco devices with little effort. The NetFlow was patented in 1996 and the first version that became available to general public around 2002 was NetFlow v5 [8], albeit Cisco newer released any official specification. The NetFlow v5 format simply specified a single set of fields that should be exported from each flow record. Figure 2.1 shows all fields that were supported by NetFlow v5. Note the lack of support for IPv6 protocol.

The NetFlow version 5 was soon obsoleted by NetFlow version 9 which remedied some of the deficiencies of the previous version. The state of NetFlow v9 is described in [9]. It allowed to define arbitrary set of fields for export using templates as shown in Figure 2.2. It also introduced support for new protocols, such as IPv6, Virtual Local Area Networks (VLAN), Multiprotocol Label Switching (MPLS), Border Gateway Protocol (BGP) or Multicast.

Other vendors created their own versions of flow exporting protocols, although they retained some level of compatibility with NetFlow. There are JFlow by Juniper, CFlow by Alcatel-Lucent, RFlow by Ericsson, and other

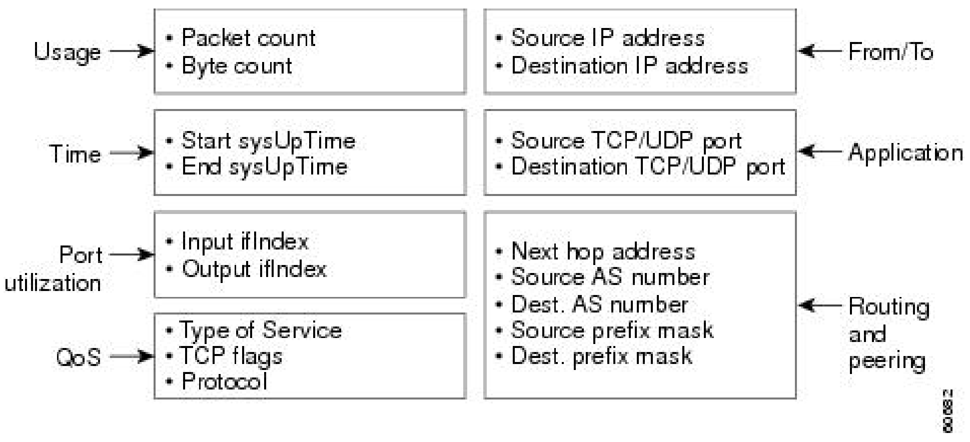

Figure 2.1: NetFlow v5 Fields

add citation

redraw

protocols. When the potential of flow monitoring for security purposes became realized [10] in 2005, more effort was devoted to extend flow records with information not directly associated with switching. Cisco presented Flexible NetFlow technology [11] in 2006 which allows to dynamically define and export new types of information, such as parts of payloads or traffic identification.

In 2001, it was clear that exporting flow information from switching devices is going to be supported by vendors. However, no standard flow export protocol existed at the time and NetFlow v5 was not yet released to general public. For that reason the IETF started IP Flow Information Export (IPFIX) WG [12]. The original charter [13] defined six specific goals for the WG:

- Define *"standard IP flow"*.
- Devise flow data encoding that support multiple levels of aggregation.
- Allow packet sampling in IP flow.
- Identify and address security and privacy concerns affecting flow data.
- Specify the transport mapping for IP flow information.
- Ensure that the flow export system is reliable.

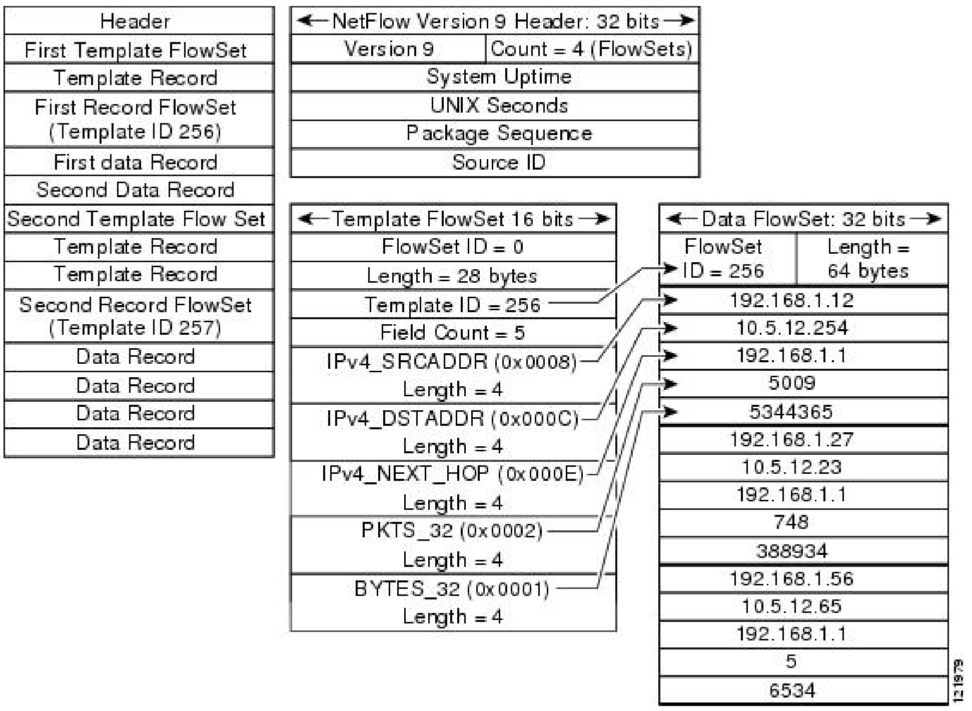

Figure 2.2: NetFlow v9 Structure Example

add citation

redraw

The charter was updated over the years to match current requirements. Several vendors were engaged in the IPFIX WG's activities, most notably Cisco, which significantly contributed from the start. The WG defined set of requirements for the IPFIX protocol [14] and evaluated existing candidate protocols [15] to decide the most suitable approach in defining the new protocol. The NetFlow v9 specification (RFC 3954) was designed with IPFIX requirements in mind [16] and was released in order to compete in this evaluation (RFC 3955). After the evaluation the NetFlow v9 was chosen as a basis of the new IPFIX protocol. For this reason, the IPFIX is sometimes called NetFlow v10 and even starts with protocol version 10 in its header. However, the IPFIX protocol supports many new features and is not completely backwards compatible with NetFlow.

The IPFIX WG did more than just design the IPFIX protocol. In the 29 RFCs submitted before its conclusion, the WG paid attention to e.g.:

- Bidirectional flow export [17]
- Architecture for IP flow information export [18]
- Reducing redundancy in flow [19]
- Definitions of Managed Objects (MIB) for IPFIX [20, 21, 22]
- IP flow mediation framework [23, 24]
- IP flow anonymization [25]
- IPFIX configuration data model [26]

The IPFIX protocol specification is described by *"Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information"* [27] which became an Internet Standard. The working group was concluded in 2014, however, IPFIX related Internet-Drafts are still being created by involved parties. Further information about IPFIX development is provided by Brownlee in [28].

2.1.2 Related Technologies

Flow monitoring is not the only network monitoring system used to gain information about network behavior. There are other technologies that can be used to monitor network traffic and that can be sometimes confused with flow monitoring. We describe sFlow [29], IETF Packet Sampling, OpenFlow and Deep Packet Inspection in the following text.

sFlow is an industry standard that is supported by number of vendors in their packet switching devices. Its initial specification was published as an Informational RFC [30] in 2001, which was the time when packet switching/routing devices with sFlow support became available. The most crucial difference from flow monitoring is that the sFlow does not actually aggregate a stream of packet into a flow record. Instead, it uses sampling to select individual packets and then exports information available about and from these packets. sFlow allows to export data from packet headers, chunks of data from packets and even parse application payloads. It also maintains interface counters and allows their regular export, which is a feature completely unrelated to flow monitoring. sFlow version 5 is the latest version and was published in 2004 [29].

In 2002 the IETF started Packet Sampling (PSAMP) Working Group [31] which was chartered to define a standard set of capabilities for network elements to sample subsets of packets by statistical and other methods [32]. The result is similar to sFlow, however, the PSAMP uses IPFIX protocol for data export [33]. The WG was concluded in 2009 after publishing four RFCs. The proposed standards include sampling and filtering techniques for IP packet selection [34], packet sampling protocol specifications [35] and information model for packet sampling export [33].

OpenFlow [36] is an open-source implementation of the Software Defined Networking (SDN) concept [37, 38]. The idea of SDN is to separate control plane and data plane of networking devices. This means that the packet forwarding rules are known only to SDN controllers. The other networking devices that process the traffic ask the controllers what to do with individual flows. After the decision is made for the first packet of the flow, a flow record is kept in the cache so that subsequent lookups do not require the controller interaction. The OpenFlow is a protocol of communication between the networking devices and the controllers. It has been shown by Yu et al. [39] that the information stored in the flow caches can be exported using the OpenFlow protocol to the controller and used for network monitoring. Although the approach to network monitoring is somewhat similar to flow monitoring on non-SDM networking devices, there are significant differences. The control traffic itself is utilized to transfer data about new and expired flow records. Therefore, configuration of flow monitoring is directly affected by configuration of SDN network and vice versa. This imposes undesirable restrictions on the flow monitoring process. Moreover, the distributed architecture of the monitoring is tightly coupled with the deployment of the network controllers. For these reasons, this thesis does not consider SDN specific flow monitoring. It should be noted, that the SDN enabled networking devices can still export valid flow data as defined by the IPFIX standard. In such a case, the SDN capabilities are irrelevant for flow monitoring purposes.

Deep Packet Inspection (DPI) is an approach to network data analysis where each packet is dissected up to and including application layer protocol (i.e. packet payload). Although this requires much greater resources than standard flow monitoring, it provides maximum information about network traffic. DPI an approach, rather than specific technology, therefore the means of packet capture and information export depend on the particular deployment. For example, sFlow uses DPI to gain information about application layer from packet payloads and exports this information as part of the sFlow protocol. Despite the DPI being diametrically different to flow monitoring, it is being integrated to flow monitoring process to provide the application visibility. This merge balances the detailed view of DPI to fast and scalable architecture of the flow monitoring. This thesis describes how the DPI is integrated to flow monitoring to create Application Flow Monitoring. Neither sFlow nor OpenFlow are not discussed any further in this work and PSAMP is only mentioned as a packet sampling protocol that can be optionally applied to flow monitoring.

2.2 Flow Definition

To be able to accurately describe the flow monitoring process, we need to have a precise definition of what a flow is. The NetFlow v9 description in [9] uses the following definition:

> *An IP Flow, also called a Flow, is defined as a set of IP packets passing an Observation Point in the network during a certain time interval. All packets that belong to a particular Flow have a set of common properties derived from the data contained in the packet and from the packet treatment at the Observation Point.*
>
> *Cisco Systems NetFlow Services Export Version 9 [9]*

The Observation Point is defined as a location where IP packets can be observed. The definition says that a flow is a set of packets within certain time span. Furthermore, the packets in a flow have a set of common properties and these properties are either derived from data contained in the packet data or from packet treatment (e.g. next hop IP address or input interface). Although this definition is quite generic, it easily covers all common flow creation techniques.

The IPFIX Protocol is an internet standard [27] with its own definition of a flow that builds upon the NetFlow v9 definition. It tries to specify what "properties derived from data contained in packet data" means and differentiates two types of data. The first are the values contained in packet headers, the second type covers the characteristics of the packet itself (e.g. packet length). The definition is as follows:

> *A Flow is defined as a set of IP packets passing an Observation Point in the network during a certain time interval. All packets belonging to a particular Flow have a set of common properties. Each property is defined as the result of applying a function to the values of:*
>
> 1. *one or more packet header fields (e.g., destination IP address), transport header fields (e.g., destination port number), or application header fields (e.g., RTP header fields [5]).*

> 2. *one or more characteristics of the packet itself (e.g., number of MPLS labels)*
> 3. *one or more fields derived from packet treatment (e.g., next hop IP address, output interface)*
>
> *A packet is defined as belonging to a Flow if it completely satisfies all the defined properties of the Flow.*
>
> *Specification of the IPFIX Protocol [27]*

Although this definition is a part of the IPFIX internet standard, there are several problems:

1. It is not clear what a *packet header* is. One interpretation is that it includes all protocol headers in the packet up to the packet payload (i.e. application layer). However, the transport header is mentioned explicitly and the example indicates that it can also mean only network layer, in which case the data link layer is completely ignored.

2. The *characteristics of the packet* are not sufficiently described. One can interpret this as anything that cannot be computed directly from the packet header fields. The example states that a number of certain types of headers is considered as part of packet characteristics. The total packet length can be also included here (it was event used as an example in the early drafts in 2002).

3. The whole IPFIX standard focuses on IP flows. However, the IPFIX export protocol can be used for other purposes as well, such as exporting MIB variables [21] or monitoring of ethernet layer [40]. Therefore the generic flow definition should allow even non-IP packets. Note that the NetFlow v9 definition of flow explicitly defines IP flows.

4. Flows using transport header fields cannot be correctly defined for fragmented IP packets, since transport layer information is present only in the first packet fragment. The NetFlow v9 definition states that the properties are *derived* from packet data, therefore, it does not rule out a use of a cache for correctly assigning or reassembling fragmented packets to correct flow. However, the use of the word *function* in the IPFIX definition limits what can be done with the values of the header fields.

In order to provide the most complete definition of flow, we must address all the above mentioned issues. The most direct solution is to start with the

NetFlow v9 definition, allow non-IP packets and be more clear about deriving data from previous packets of the same flow which is used for correct handling of the packet fragmentation. Therefore, the definition used in this thesis is as follows:

> **Definition 2.1**
> *A* flow *is defined as a sequence of packets passing an* observation point *in the network during a certain time interval. All packets that belong to a particular* flow *have a set of common properties derived from the data contained in the packet, previous packets of the same* flow*, and from the packet treatment at the* observation point*.*

There are two more terms connected to flow that need to be defined: *flow keys* and *flow records*. The IPFIX definition of the Flow Key needs to be adapted to our definition of flow. We can conveniently shorten the definition to the following:

> **Definition 2.2**
> *Each of the common properties that is used to specify a* flow *is called a* flow key*.*

A flow record is basically a tuple containing flow keys and other properties measured for the flow. The following definition reflects that:

> **Definition 2.3**
> *A* flow record *is a tuple describing particular* flow *containing values of:*
>
> 1. *all* flow keys *used to specify the* flow*,*
> 2. *other properties of the* flow *derived from:*
>   (a) *data contained in the packets of the* flow*,*
>   (b) *the packet treatment of the* flow *at the* observation point*.*

To make the definitions above more clear, we now give an example of concrete properties that might be contained in a flow record. Table 2.1 shows examples of flow record properties that can be derived from packet data and packet treatment. The properties can be aggregated when the derived value differs between individual packets of the flow or where counters such as number of packets are involved. Summary function is usually applied to number of bytes of of each packet, TCP flags are aggregated using logical

	Aggregated properties	**Non-aggregated properties**
Packet data	Number of bytes TCP flags Time to Live	Source IP address Destination port Transport protocol
Packet treatment	Number of packets Flow start timestamp	Input interface number Next-Hop IP address

Table 2.1: Examples of Flow Properties

OR function, flow start timestamp is derived using minimum function on each packet timestamp. The non-aggregated properties may be used as flow keys.

The Definition 2.1 states what the flow is. However, although we tried to be as explicit as possible, the definition is informal and therefore subject to different interpretations. For this reason we now provide a formal definition of flow, which not only refines the informal definition, but also provides a guide to construction of the flows.

Definition 2.4
Let P be a set of all packets. Let T be a set of packet treatment information. We define a set of extended packets

$$\widehat{P} = P \times T,$$

so that $\widehat{p} \in \widehat{\mathcal{P}}$ denotes a packet p together with its packet treatment information. Let $\mathbb{S}$ be a set of indexes of packets observed at an observation point*:*

$$\mathbb{S} = \{1, \ldots, n\} \vee \mathbb{N},$$

where $n \in \mathbb{N}$ is the number of observed packets when the number is finite.

We denote sequence of packets and extended packets observed at an observation point *respectively:*

$$\mathcal{P} = (p_i)_{i \in \mathbb{S}},\ p_i \in P,$$
$$\widehat{\mathcal{P}} = (\widehat{p}_i)_{i \in \mathbb{S}},\ \widehat{p}_i \in \widehat{P}.$$

Both sequences are of size $|\mathbb{S}|$.

Let us now define a *flow selection function* φ which takes a sequence of extended packets and a new extended packet and decides whether they form a flow. We will use this function to determine whether a newly observed packet belongs to an existing flow.

Definition 2.5
Let $\widehat{P}^*$ be a set of all finite sequences of extended packets, $\widehat{P}$ be a set of extended packets. We say that a function of type

$$\varphi : \widehat{P}^* \times \widehat{P} \rightarrow \{true, false\}$$

is a flow selection function.

Before we give a formal definition of a flow, we provide the following intuition for our definition. A flow $\mathcal{F}$ is a sequence of packets defined by a sequence of extended packets with indexes in $\mathbb{S}$ and a *flow selection function* φ. We require whether each packet belongs to the flows is determined by all previous packets of that flow, therefore we construct the flow by induction as described in Algorithm 2.1.

```
1: Denote $\mathbb{I}$ the set of indexes of packets that belong to the flow $\mathcal{F}$
2: Start with $\mathbb{I} = \emptyset$
3: while An index $k$ of the first extended packet $\widehat{p}_k$ for which
   $\varphi((\widehat{p}_n)_{n\in\mathbb{I}}, \widehat{p}_\alpha) = true$ exists do
4:     Add $k$ to $\mathbb{I}$
5: end while
6: The flow $\mathcal{F}$ is a sequence of packets with indexes from $\mathbb{I}$
```

Algorithm 2.1: Construction of a flow

We now define set $\mathbb{I}$ of indexes from $(\widehat{p}_i)$ selected using *flow selection function* φ, and flow $\mathcal{F}$ so that it conforms with the Definition 2.1 as follows:

Definition 2.6
Let $(p_i)_{i\in S}, (\widehat{p}_i)_{i\in S}, S \subseteq \mathbb{N}$ be (possibly finite) mutually corresponding sequences of packets and extended packets respectively, φ a flow selection function.
We define a flow index set $\mathbb{I} = \mathbb{I}\left((\widehat{p}_i)_{i\in S}, \varphi\right)$ *as*

$$\mathbb{I} = \lim_{i\to\infty} J_i, \textit{ where } J_i \textit{ is defined inductively over } i \in \mathbb{N} \textit{ as:}$$

$$J_i = \begin{cases} \{\min\{\alpha \in S \mid \varphi(\widehat{p}_\alpha) = true\}\} & \textit{for } i = 1, \\ J_{i-1} \cup \{\min\{\alpha \in S \mid \alpha > \sup(J_{i-1}), \\ \quad \varphi((\widehat{p}_n)_{n\in J_{i-1}}, \widehat{p}_\alpha) = true\}\} & \textit{for } i > 1. \end{cases}$$

Finally, we define flow $\mathcal{F} = \mathcal{F}\left((p_i)_{i\in S}, \mathbb{I}\right)$ as:

$$\mathcal{F} = (p_i)_{i\in\mathbb{I}}, \, p_i \in \mathcal{P}.$$

Since we need the $\min$ function to be defined for empty set (the cases where no flow is defined and where we have already added all possible indexes from $\mathbb{S}$), we define

$$\{\min\ \emptyset\} = \emptyset$$

The Definition 2.6 of flow creates single flow for a sequence of extended packets $\widehat{\mathcal{P}}$ and a *flow selection function* φ. The flow $\mathcal{F}$ is selected based on the first extended packet accepted by φ. Since we naturally expect that every packet is part of only a single flow, we can construct a sequence of flows $(\mathcal{F}_i)_{i\in\mathbb{N}}$ by induction as described in Algorithm 2.2.

```
1: Denote $S_1 = \mathbb{S}$
2: Set counter $i = 1$
3: repeat
4:     Apply the flow selection function $\varphi$ to extended packets with indexes in $S_i$
5:     Denote indexes of matching extended packets $\mathbb{I}_i$
6:     Flow $\mathcal{F}_i$ is a sequence of packets with indexes from $\mathbb{I}_i$
7:     Remove indexes in $\mathbb{I}_i$ from $S_i$, denote the new sequence $S_{i+1}$
8:     Increment counter $i = i + 1$
9: until $\mathcal{F}_i$ is empty
```

Algorithm 2.2: Construction of a sequence of flows

Let us now provide a more formal definition of a sequence of flows $(\mathcal{F}_i)_{i\in\mathbb{N}}$.

Definition 2.7
Let $\mathcal{P}, \widehat{\mathcal{P}}$ be sequences of packets and extended packets respectively, φ a flow selection function. *We define the sequence $(\mathcal{F}_i)_{i\in\mathbb{N}}$ of flows inductively:*

$$\begin{aligned}
&\mathcal{F}_i = \mathcal{F}_i\left((p_j)_{j\in S_i},\, \mathbb{I}_i\right), \textit{where}\\
&S_1 = \mathbb{S},\\
&S_i = S_{i-1} \setminus \mathbb{I}_{i-1},\\
&\mathbb{I}_i = \mathbb{I}_i\left((\widehat{p}_j)_{j\in S_i},\, \varphi\right).
\end{aligned}$$

The Definition 2.7 provides a guide to constructing a sequence of flow records. The procedure can be easily modified to run in real time so that each newly observed extended packet can be added to the appropriate flow. Since we know that every packet is a part of only a single flow, we apply the *flow selection function* φ to each existing flow (enriched by packet treatment information) and the new extended packet and add the packet to the

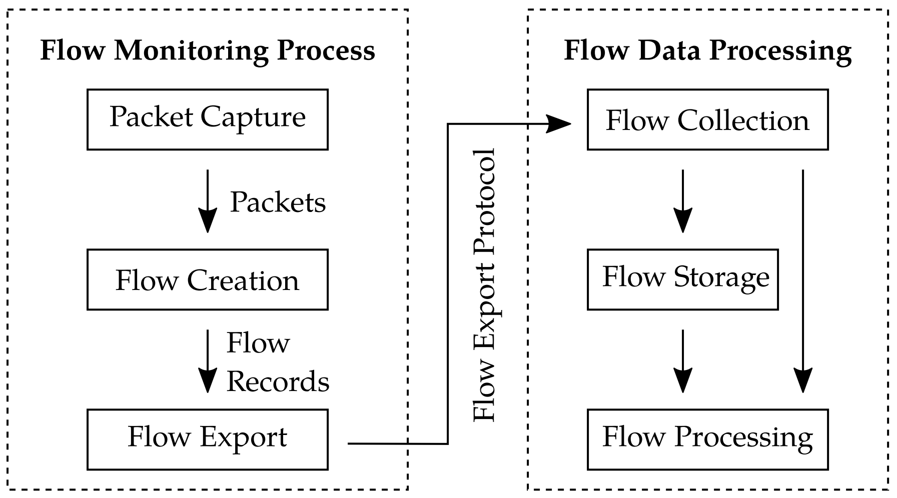

Figure 2.3: High Level Flow Monitoring Schema

first flow that matches. If none of the existing flow matches, we apply the function φ to this packet only and start a new flow if necessary.

We can see that the flow creation process significantly depends upon the implementation of the *flow selection function*. We will discuss the common implementations in the Subsection 2.4.3.

2.3 Flow Monitoring Architecture

Deployment of flow monitoring on a network requires several steps: Capturing packets at one or more observation points, assigning packets to flows, creating and exporting flow records for the flows, and finally collecting, storing, and processing of the exported flow records. The Figure 2.3 shows a high level overview of the whole process. Flow monitoring process encompasses packet capture, flow creation, and creation and export of flow records. Flow data processing comprises of flow record collection, storage, and further processing. Note that the flow records can be processes directly without storing. This approach is called *stream processing*. It is also possible to manipulate the flow records in transition between the export and collection. The IPFIX working group specified a framework called IP Flow Information Export Mediation [24] which describes this process. However, description of this process is outside the scope of this thesis.

This rest of section explains basic terminology and components of the flow monitoring architecture and describes the most commonly deployed flow monitoring architectures.

2.3.1 Terminology

The IPFIX working group published several documents where the architecture for IP Flow Information Export is described [18, 24]. However, the terminology used in these documents is not commonly used by the flow monitoring community and some of the terms have different meaning depending on a context. We start by presenting the IPFIX reference model as described in RFC5470 [18], which can also be used to describe generic flow monitoring architecture. Then we identify the terms that are often used with different meaning and explain how these terms are used throughout this thesis.

The following definitions are taken from RFC 7011 [27]:

Observation Point

An Observation Point is a location in the network where packets can be observed. Examples include a line to which a probe is attached; a shared medium, such as an Ethernet-based LAN; a single port of a router; or a set of interfaces (physical or logical) of a router.

Note that every Observation Point is associated with an Observation Domain (defined below) and that one Observation Point may be a superset of several other Observation Points. For example, one Observation Point can be an entire line card. That would be the superset of the individual Observation Points at the line card's interfaces.

Observation Domain

An Observation Domain is the largest set of Observation Points for which Flow information can be aggregated by a Metering Process. For example, a router line card may be an Observation Domain if it is composed of several interfaces, each of which is an Observation Point. In the IPFIX Message it generates, the Observation Domain includes its Observation Domain ID, which is unique per Exporting Process. That way, the Collecting Process can identify the specific Observation Domain from the Exporter that sends the IPFIX Messages. Every Observation Point is associated with an Observation Domain. It is RECOMMENDED that Observation Domain IDs also be unique per IPFIX Device.

Packet Treatment

"Packet Treatment" refers to action(s) performed on a packet by a forwarding device or other middlebox, including forwarding, dropping, delaying for traffic-shaping purposes, etc.

Metering Process

The Metering Process generates Flow Records. Inputs to the process are packet headers, characteristics, and Packet Treatment observed at one or more Observation Points.

The Metering Process consists of a set of functions that includes packet header capturing, timestamping, sampling, classifying, and maintaining Flow Records.

The maintenance of Flow Records may include creating new records, updating existing ones, computing Flow statistics, deriving further Flow properties, detecting Flow expiration, passing Flow Records to the Exporting Process, and deleting Flow Records.

Exporting Process

The Exporting Process sends IPFIX Messages to one or more Collecting Processes. The Flow Records in the Messages are generated by one or more Metering Processes.

Exporter

A device that hosts one or more Exporting Processes is termed an Exporter.

IPFIX Device

An IPFIX Device hosts at least one Exporting Process. It may host further Exporting Processes as well as arbitrary numbers of Observation Points and Metering Processes.

Collecting Process

A Collecting Process receives IPFIX Messages from one or more Exporting Processes. The Collecting Process might process or store Flow Records received within these Messages, but such actions are out of scope for this document.

Collector

A device that hosts one or more Collecting Processes is termed a Collector.

Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information [27]

IPFIX vs thesis
- Flow exporter, probe, flow meter vs IPFIX terminology (metering process, sampling, filtering, exporting process)
- IPFIX as IP Flow Information eXport standard (describes architecture as well) - Here or in/after architecture?

Observation Point	Logical collection of packet sources??? TODO
Observation Domain	Logical collection of Obsrv Points
Packet Treatment	Packet processing on switch/router
Metering Process	Exporter
Exporting Process	Exporter (sw/hw)
Exporter	-
IPFIX Device	Probe, Switch, etc
Collecting Process	Collector
Collector	Collector

2.3.2 Monitoring Configuration

- High level overview of the components
- Dedicated probe: SPAN, TAP
- Switch/Router device
- Monitoring configurations (SPAN - no direction information)
- Where to measure: inside network, network border, combination (pitfalls), NAT devices (lemo - postprocessing/mediator)

2.4 Flow Monitoring Process

Part is already written in next chapter under Creating Application Flow
Put generic stuff here and keep application related information there

- Packet observation
- - Probe location (wired, wireless, virtual)
- - Port mirroring vs in-line (tap)
- - Technologies: PF_RING, PF_RING ZC, Linux NAPI, FPGA (DPDK, PF_RING, DNA/Libzero)
- Explain that we do not perform sampling and filtering (filtering only in NIC for specific purposes or on exporter for testing, etc).
- Flow metering (caches, ...), biflows
- Flow expiration/termination - Flow export

2.4.1 Packet Capture

2.4.2 Packet Parsing

2.4.3 Flow Creation

2.4.4 Flow Export

Same section in chapter 3 (app flow monitoring), keep it consistent

- Start with IPFIX, show that there are other options
- Transport protocols
- IPFIX protocol, json, ...
- TCP, UDP, SCTP pros and cons (TCP can easily block too much, message must be dropped somewhere)
- Message lengths, MTU, packet fragmentation

2.5 Flow Data Processing

2.5.1 Flow Collection

2.5.2 Flow Storage

2.5.3 Flow Processing

- Flow collectors (https://github.com/VerizonDigital/vflow)
- Flow data storage (data formats)
- Flow data analysis (query, stream vs static)

2.6 Common Issues

- What can go wrong with flow measurement (new encapsulation protocols, data loss)
- Performance (link congestion, collector congestion, exporter congestion - TCP)
- Misconfiguration (fw, ports, packet sizes, missing and/or incompatible IE definitions)
- Put packet fragmentation here?

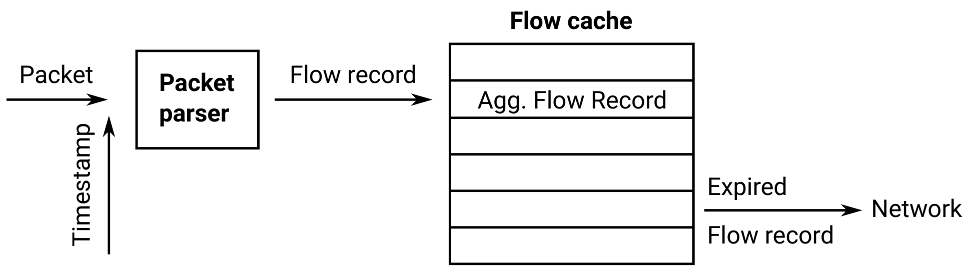

Figure 3.1: A Flow Exporter Schema

3 Application Flow Monitoring

- Cisco Application Visibility and Control
- `http://www.cisco.com/c/en/us/products/routers/avc-control.html`
- `http://www.cisco.com/c/dam/en/us/solutions/collateral/enterprise-networks/unified-wan-services/at_a_glance_c45-649117.pdf`

- Network-Based Application Recognition (NBAR)
- `http://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/network-based-application-recognition-nbar/product_bulletin_c25-627831.html`

3.1 Creating Application Flow

Taken from: Application Flow Monitoring Challenges

TODO: vyporadat se s opakujicimi se hlavickami (HTTP), doplnit definici flow z predchozi kapitoly
- Flow expiration/termination vs app flow splitting

The application flow monitoring is a complex process. Packets need to be received from the network, transported to computer memory (RAM), parsed, aggregated to flow records based on the parsed information, and exported to flow collectors for further processing. The process is depicted in Figure 3.1. This section describes the whole flow measurement process, points out where it is affected by the application processing and introduces some of the related challenges.

3.1.1 Packet Reception

The packet reception ensures that data from the network are made available for processing in the software. Standard network interface cards (NIC) can be used, as well as specialized hardware accelerated cards. Either the NIC, its driver, or the data processing application must assign a timestamp to each packet. More information about packet capture can be found for example in the work of García-Dorado et al. [41]. The packet reception is unchanged for application flow monitoring unless the NIC processes the application layer. This scenario is discussed later in Hardware Accelerated Techniques section.

3.1.2 Packet Parsing

The basic task of the packet parsing is to extract connection attributes such as IP addresses, transport protocol, and ports to determine which flow the packet belongs to. Moreover, it obtains additional information of interest, especially from application protocols. The parser must be resilient to malformed packets and unknown protocols while supporting a wide range of existing network and application protocols.

Link, network, and transport headers of IP packets follow a strictly defined structure so that the network devices such as routers can process the packets swiftly. However, application layer protocols often rely on connections being established between compatible endpoints. For this reason, application protocol identification is a difficult task. A lot of attention has been dedicated to research of application identification in network traffic in the past, for example in the work of Bujlow et al. [42], however, not every approach is suitable for the flow monitoring scenario.

With increasing deployment of encryption for all kinds of communication, the task of application flow monitoring becomes more difficult. Without access to application payload, the amount of information that can be extracted from the traffic is diminished. However, there are statistical and machine learning methods that are able to recognize specific applications even in encrypted traffic with high accuracy. Moreover, useful information such as a version of encryption protocol, certificates, or supported cipher suites, can be extracted from encrypted traffic. This information can be used to identify malicious encrypted traffic. The possibilities of processing and analysis of encrypted traffic were surveyed by the authors of [43].

3.1.3 Flow Aggregation

After each packet is parsed, the extracted information is stored in a flow record. Flow records for the same connection are aggregated in a flow cache. A connection is usually identified by several key features such as IP addresses, transport protocol, and ports. Each flow record has two timestamps: flow start timestamp is the time that the first packet of the flow was observed, end timestamp is the timestamp of the last packet.

There are several reasons why a flow record can exit the flow cache. When no new packets belonging to the flow arrive for a time interval called inactive timeout, or when the flow has been in the cache for a time interval called active timeout, the flow is expired and removed from the flow cache. The active timeout is usually much longer than the inactive one. For example, Cisco IOS flow cache has the default of 30 minutes for the active timeout and 15 seconds for the inactive one[1]. Another reason for flow expiration that is used in practice is when TCP protocol packet with RST or FIN flag is encountered. Flow expiration setting can significantly influence not only the number of generated flow records, as shown by the authors of [44], but also further flow processing and various flow-based detection methods.

Application protocol measurement may require flow record to be expired early. For example, when HTTP protocol supports pipelining, multiple requests and responses can be carried out over a single connection. When it is desirable to keep track of each request/response pair, existing flow record might be exported when a new request is encountered on the same connection. Therefore, application flow monitoring also affects the number of generated flow records, which needs to be taken into consideration during further processing of the flow records.

TODO: keeping application data from expired flow records (long HTTP video streaming)

Another impact of the application flow monitoring on the flow aggregation is the increased size of flow records. The information extracted from application protocols can be quite large in comparison to network and transport layers lengths. While the typical IPv4 header length is 20 bytes, typical TCP header length is 32 bytes, the HTTP URL can easily be several hundred bytes long. Therefore, the length of flow records of application flow monitoring is several times larger than without the application layer. There are two main negative impacts of such large flow records. Firstly, the flow cache

1. `http://www.cisco.com/c/en/us/td/docs/ios/fnetflow/command/reference/fnf_book/fnf_01.html`

might require much more RAM than standard flow monitoring flow cache. Secondly, even if the cache fits into RAM, it degrades the performance of the memory accesses because data locality is decreased and a CPU experiences more cache misses. For these reasons, it must be carefully considered which information is placed in each flow record and how it is encoded.

Little attention has been given to the measurement of traffic with fragmented packets. Although packet fragmentation poses a problem for standard flow measurement as well, the application measurement is especially affected. Let us consider what happens when an IPv4 packet is fragmented (the problem is very similar for IPv6). Figure 3.2 illustrates this scenario. When the packet K is fragmented, its payload is distributed between several different packets $(K\#1, K\#2, \ldots, K\#N_K)$. Only the first packet $(K\#1)$ contains the transport header, others only carry a flag identifying the fragment and offset of the data in the original packet. When a flow record is created for the first part of the fragmented packet, it contains IP addresses, transport layer information (protocol and ports), and the first part of application payload. However, subsequent parts contain only information about IP addresses and consecutive parts of the application payload. The transport level information was already sent in the first fragment. Therefore, flow records created from the subsequent fragments are not aggregated with the first fragment and application parsers are seeing only incomplete payloads. Moreover, flow records created from subsequent fragments from different connections between the same hosts are aggregated together. To resolve this problem, some kind of packet reassembly must happen in the flow cache. One possible solution that has been successfully deployed in practice is to create temporary flow records for fragmented packets based on fragmentation identifiers instead of transport layer information. Therefore, each fragmented packet has its own temporary flow record. These flow records have shorter timeouts, so when all fragments are received, the temporary flow record can be reinserted into the flow cache with proper transport layer information, which is now available since it was present in the first fragment.

3.1.4 Flow Export

Same section in chapter 2, keep it consistent

After the flow record is expired from the flow cache, it is encoded in suitable protocol and sent to a flow collector for further processing. The most widely used protocol for flow records is CISCO NetFlow, which is being replaced by IETF defined IPFIX protocol nowadays. Only the IPFIX protocol

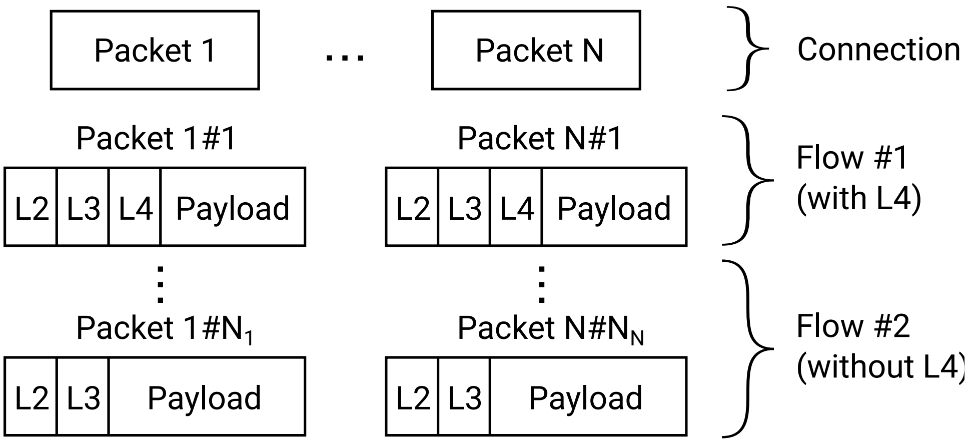

Figure 3.2: Flow measurement of a fragmented connection.

is flexible enough to support application flow monitoring. Although CISCO has added some application elements to the NetFlow protocol as well, no third party application elements are supported.

Several issues might be encountered with flow export when application flow monitoring is applied. First, due to the larger amount of exported data, the link to a flow collector might be congested. We have experienced this issue when application flow records from a 10G link were exported simultaneously to several collectors over old 100 Mb/s management interface. The solution is either to simply upgrade to 1 Gb/s management interface or, better, to reduce the number of targets of the export and distribute the flows as necessary through replicating proxy in the location of the collectors. The latter solution saves network bandwidth used for monitoring purposes and should be preferred if possible.

Another issue that might be encountered due to large flow record is that single record might be larger than MTU of the management network interface on the flow exporting device. In such a case UDP transport protocol, which is still widely used for flow export, cannot be used without fragmenting IP packets. The fragmentation might cause a performance problem for the flow collector which has to reassemble the packets. It also increases the probability of data loss as single lost fragment invalidates the whole message.

3.2 Application Flow Benefits

Papers using application flows

3.3 Conclusions

3.4 Relevant Publications

Do it like rick and have it in the intro to each chapter?

4 Application Flow Monitoring Performance

Text below is mostly from Challenges of Application Flow Monitoring.

TODO: Howto measure flow monitoring performance - skoro samostatná kapitola

TODO: biflow

TODO: Include more details

TODO: Give a background about high-speed packet capture, lucaderi, etc.

TODO: Put in papers from IM2015

TODO: Title is Flow Monitoring Performance, split to application and classic

Speedup by ommiting SSL/TLS processing for most application plugins

4.1 Accelerating Application Flow Monitoring

We have shown that there are many different challenges when building application flow monitoring system. Most of the described processes are performance sensitive, especially packet reception, packet parsing, and flow aggregation. To achieve application flow monitoring throughput of tens of gigabits per second, several optimization and acceleration techniques can be applied. This section focuses on these techniques. Note that we avoid the use of sampling since it degrades the quality of exported data, as shown by the authors of [45]. Although we describe acceleration techniques, their design, and main ideas, we will not go into details about their evaluation and testing, which is out of the scope of this article.

It has been shown by the authors of [46] that standard flow monitoring can achieve more than 100 Gb/s throughput on a commodity CPU when hardware accelerated NICs are used. The work of Kekely et al. [47] shows that it is possible to utilize these NICs to achieve 100 Gb/s throughput even for application flow monitoring. We discuss both hardware accelerated techniques and software optimizations in this section. Table 4.1 gives an overview of the discussed techniques.

4.1.1 Hardware Accelerated Techniques

A field-programmable gate array (FPGA) is usually used in hardware accelerated network interface cards. It allows to parse packet headers and perform other tasks such as packet trimming or computing hashes to identify

Hardware acceleration	**Software acceleration**
Multiple reception buffers	Multithreading
Packet trimming	NUMA awareness
Packet header preprocessing	Flow state in parsers
Flow processing offloading	Flow cache design
Application identification	Flow cache timeouts

Table 4.1: Flow Acceleration Techniques

flows in the NICs. It usually supports transferring data to multiple buffers in RAM so that the software can efficiently use a multi-threading model. There are several manufacturers that provide such NICs, such as Napatech, Myricom, Mellanox Technologies, or Netcope Technologies. NICs can support hardware acceleration even without an FPGA chip, however, the capabilities are usually more limited and cannot be extended after the card is produced. For example, Intel produces many NICs without an FPGA chip that provide basic hardware acceleration such as packet hashing and transport to multiple RAM buffers.

By transferring data to multiple buffers in RAM, the NIC allows multiple CPU cores to process the data without any kind of locking mechanism. The data parsing is usually the most CPU intensive part of the application flow creation process, therefore it is very desirable to parallelize the computation. It is often necessary for the application packet parsing process to see multiple packets from the same flow. Therefore, it is desirable to store packets belonging to single flow to a single buffer in RAM, so that they are not processed by multiple different threads. To achieve this, the NIC must be able to correctly categorize packets belonging to the same flow. However, it is not necessary for the NIC to differentiate between every flow record. For example, when eight buffers are used, each flow must be sent to exactly one of the buffers. The easiest way to achieve this is to compute a three-bit hash from some of the flow-defining fields in the packet headers. IP addresses and transport layer ports are often used for this purpose, however, most manufacturers do not realize that this works incorrectly for fragmented packets. Therefore, it is better, in most scenarios, to use only IP addresses to distribute the flows to different buffers. The performance improvement achieved when using multiple buffers is directly related to the number of buffers. However, too many buffers introduce a high load on a memory controller causing the performance to be degraded. It is best to experiment with different numbers of buffers to find an optimal value for the target system. We have achieved the best results with 8-16 buffers, depending on the specific scenario and the number of CPU cores.

Memory throughput can easily become a bottleneck when data from 100 Gb/s link are sent to RAM. One of the options to reduce the amount of processed data is to trim the received packets. Flow measurement without application protocols requires only packet headers up to the transport layer. Capturing the first 100 bytes of each packet is usually enough to contain various MPLS, VLAN, Ethernet, IPv6, and TCP headers. The rest of the packet can be discarded. The performance improvement achieved by this method depends on the average length of processed packets. However, application flow measurement requires also the packet payload. The amount of data that can be discarded varies depending on the measured application protocols. Unless the NIC has additional knowledge about measured traffic, it should not trim the packets at all for application flow measurement.

More radical way to save memory bandwidth and CPU cycles is to let the NIC extract the information required for flow monitoring and send only a special message with this information to the RAM. This process requires very specialized FPGA-based cards, such as COMBO series from CESNET. The performance gain can be quite high, as shown in [46]. However, this method is not applicable for application flow measurement since application layer parsing is too complex and dynamic to be fully handled in the FPGA.

Packet trimming and packet parsing in the NIC are not usable for application flow monitoring. The main reason is that application layer parsing needs to be done in software. However, most packets do not carry application protocol headers. These packets can be processed by NIC and only extracted information transferred to software. Such a solution is proposed in [47] and is called Software Defined Monitoring (SDM). It is based on offloading of heavy flows to the NIC. The important information from application layer is usually transferred in first N packets, where N can be expected to be lower than 20 in most cases. Therefore, after the software processing encounters the Nth packet, it instructs the NIC to process the rest of the flow. Only aggregated information about the flow is sent from NIC to software. The flow cache in the NIC is rather limited in comparison to the amount of RAM available in the software. However, only heavy flows need to be kept in this cache and it can be expired more often to reduce memory requirements. The authors show that 85 percent of traffic is observed in five percent of heavy flows. Therefore, the benefit of SDM has been shown to be an aggregation of 85 percent of packets to flow records in the NIC. This significant acceleration allows application flows to be monitored on 100 Gb/s traffic.

However, there is a downside to the SDM as well. Once a flow is offloaded to NIC, software parsers cannot observe further payloads containing subsequent application headers. For example, in the case of HTTP protocol the HTTP pipelining cannot be detected and information about further requests and responses in the same connection is lost.

Since only a small portion of all packets carry important application protocol information, it is beneficial to recognize such packets in the NIC and mark them for further processing in software. As the NIC cannot do complex application header processing, only a simple mechanism, such as pattern matching, can be utilized for packet classification. The work of the WAND Research Group [48] shows that it is possible to achieve reasonable traffic classification accuracy using only the beginning of a packet payload. Once the packets carrying application protocol headers are marked, the software parsers can process only these important packets. Moreover, other packets can be trimmed or parsed in the NIC and only important information can be passed for processing in the software. Packet classification can also benefit the SDM. When a flow is offloaded to NIC and a packet with application protocol header is matched, the processing of such a flow can be returned to the software. This approach would efficiently solve the abovementioned problem of HTTP pipelining. The accuracy and usability of pattern matching for packet classification depend on target application protocols and resource limitation of the NIC as well. Further research is needed to determine whether this method can be used for real world application flow monitoring.

4.1.2 Acceleration in Software

The performance of flow monitoring can be greatly enhanced by offloading as much of the necessary packet processing as possible to the NIC. However, to build a high-performance flow monitoring solution the software processing must be carefully tuned as well. Moreover, the FPGA-based NICs are much more expensive than commodity NICs and cannot always be deployed. This section focuses on flow monitoring optimizations that can be achieved by careful software design and system configuration.

The development of modern CPUs has a tendency to substitute raw power (frequency) by a higher number of CPU cores. To fully utilize the potential of a multi-core CPUs and achieve high flow monitoring performance, it is necessary to create multithreaded flow monitoring software. For NICs that support storing data in multiple buffers in RAM, the solution is to have different threads process the packets from different buffers. The resulting

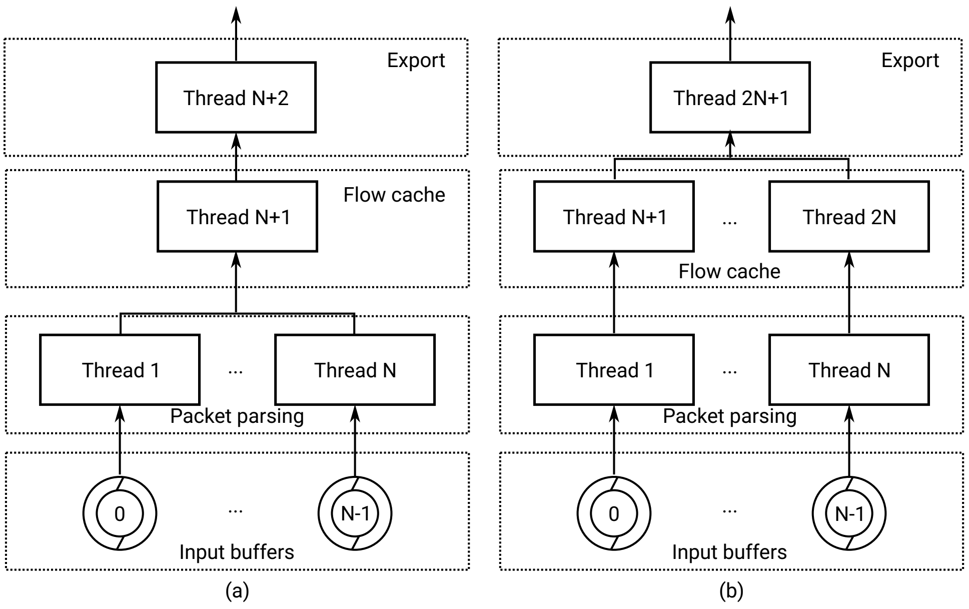

Figure 4.1: Multithreaded flow measurement with separate flow cache: a) single flow cache; b) multiple flow caches.

flow records are aggregated either in a single flow cache or in multiple flow caches, one per each thread, as shown in Figure 4.1. In either case, it must be ensured that there is a single thread used for exporting the resulting flow records where the data from multiple processing threads are merged. The latter approach is more effective as it clearly prevents the flow cache to become a point of contention. However, when there are multiple flow caches, it is necessary to ensure that each flow is aggregated in a single flow cache. This is ensured by having the NIC correctly distribute the packets to the buffers according to their association with their respective flow. Special care needs to be taken when the application protocol parsers need to process reverse flows as well. Both directions of the communication must be processed by the same thread. Using this approach, $N+2$ threads in case the first case or $2N+1$ threads in the second case can be utilized, where N is the number of buffers and the last thread is used for flow export.

Separating packet parsing from flow cache management increases performance, however, processing application protocols may require a state of the connection to be kept. In such a case the flow cache record must be made available to the packet processing thread, which results in a use of synchronization primitives and overall performance decrease. One possible solution to this problem is to keep a different, smaller cache for chosen flow records directly in the packet processing thread. The only communication between the packet processing thread and the flow cache thread is a one-way passing of flow records, which can be done very effectively.

When a server with multiple CPUs is used, it is necessary to take care to assign each thread to correct CPU core. When the data is uploaded to RAM physically connected to different CPU than the packet processing core is running on, there is a performance hit for accessing the local memory of that CPU.

It is not effective to try every available application header parser on every packet to see which one is able to process it. When a packet from a flow was already matched by some application parser, it will usually not be matched by others. Therefore, by keeping the information about which flow is to be processed by each parser, most of the unnecessary and possibly expensive calls to application parsers can be eliminated. Moreover, when the flow is yet unmatched by any application parser, it is best to execute the application protocol parsers from the most common to the least common protocol.

One of the most performance critical parts of any flow measurement software is a flow cache. The cache needs to be designed to hold hundreds of thousands of flow records, do fast inserts, updates, and manage record expiration after active and inactive timeouts. It also needs to be robust and resilient enough to handle excessive workloads during DDoS attacks [49]. The flow cache design has been studied in the literature, for example in[50, 51]. Although authors propose to use various data structures such as linked lists, trees, or multidimensional hashing table, our experience shows that the simplest solution is the best. The flow cache has to maintain data locality to make good use of CPU caches, therefore the dynamic structures do not perform as well as a simple hash table.

Flow cache inactive timeout expiration has a large impact on the performance, as shown in [44, 52]. The flow cache needs to be checked periodically to find and expire inactive records. However, doing the periodic checking in a separate thread requires extensive flow cache locking, which hinders the performance. Therefore, it is more efficient to dedicate part of the processing time of the flow cache thread itself to search for the inactive records. Carefully balancing the flow cache management tasks is a complex problem which offers a considerable potential for further research.

There are many other optimizations that can be performed to increase application flow monitoring performance, such as efficient flow key computation, processing packets in batches, ensuring CPU cache line alignment of flow records and so on. However, they are mostly a code micro-optimization no different from fine-tuning any other high-performance application and are out of the scope of this article.

4.2 Conclusions

4.3 Relevant Publications

5 Measurement of Encrypted Traffic

5.1 Conclusions

5.2 Relevant Publications

6 Next Generation Flow

6.1 EventFlow

6.2 Flexible Flow Collector

- IPFIXcol
- Collector design
- Problems with IPFIX protocol processing (Projít s Lukášem Hutákem)
- Flexible flow data storage (db vs nfdump vs FastBit vs new format?)
- Mediator: fix intro in chapter 2: section Flow Monitoring Architecture if described here

- Take a look at https://github.com/VerizonDigital/vflow

6.3 Conclusions

6.4 Relevant Publications

- AIMS 2012 ipfixcol design
- IM 2013 fbitdump vs nfdump comparison
- NOMS 2015 EventFlow

7 Conclusions

7.1 Answers to Research Questions

7.2 Further Research

Bibliography

1. HOFSTEDE, Rick; ČELEDA, Pavel; TRAMMELL, Brian; DRAGO, Idilio; SADRE, Ramin; SPEROTTO, Anna; PRAS, Aiko. Flow Monitoring Explained: From Packet Capture to Data Analysis with NetFlow and IPFIX. *Communications Surveys Tutorials, IEEE*. 2014, vol. PP, no. 99, pp. 2037–2064. ISSN 1553-877X. Available from DOI: `10.1109/COMST.2014.2321898`.
2. CEJKA, Tomas; BARTOS, Vaclav; TRUXA, Lukas; KUBATOVA, Hana. Using Application-Aware Flow Monitoring for SIP Fraud Detection. In: LATRÉ, Steven; CHARALAMBIDES, Marinos; FRANÇOIS, Jérôme; SCHMITT, Corinna; STILLER, Burkhard (eds.). *Intelligent Mechanisms for Network Configuration and Security: 9th IFIP WG 6.6 International Conference on Autonomous Infrastructure, Management, and Security, AIMS 2015, Ghent, Belgium, June 22-25, 2015. Proceedings*. Cham: Springer International Publishing, 2015, pp. 87–99. ISBN 978-3-319-20034-7. Available from DOI: `10.1007/978-3-319-20034-7_10`.
3. HUSÁK, Martin; VELAN, Petr; VYKOPAL, Jan. Security Monitoring of HTTP Traffic Using Extended Flows. In: *2015 10th International Conference on Availability, Reliability and Security*. 2015, pp. 258–265. Available from DOI: `10.1109/ARES.2015.42`.
4. MILLS, C.; HIRSH, D.; RUTH, G.R. *Internet Accounting: Background* [RFC 1272 (Informational)]. Fremont, CA, USA: RFC Editor, 1991. Internet Request for Comments, no. 1272. ISSN 2070-1721. Available from DOI: `10.17487/RFC1272`.
5. FARRELL, S.; TSCHOFENIG, H. *Pervasive Monitoring Is an Attack* [RFC 7258 (Best Current Practice)]. Fremont, CA, USA: RFC Editor, 2014. Internet Request for Comments, no. 7258. ISSN 2070-1721. Available from DOI: `10.17487/RFC7258`.
6. CLAFFY, Kimberly C.; BRAUN, Hans-Werner; POLYZOS, George C. A Parameterizable Methodology for Internet Traffic Flow Profiling. *IEEE Journal on Selected Areas in Communications*. 1995, vol. 13, no. 8, pp. 1481–1494. ISSN 0733-8716. Available from DOI: `10.1109/49.464717`.
7. BROWNLEE, N.; MILLS, C.; RUTH, G. *Traffic Flow Measurement: Architecture* [RFC 2722 (Informational)]. Fremont, CA, USA: RFC Editor, 1999. Internet Request for Comments, no. 2722. ISSN 2070-1721. Available from DOI: `10.17487/RFC2722`.

8. CISCO SYSTEMS, INC., SAN JOSE, CA AND USA. *NetFlow Services Solutions Guide* [online]. 2007 [visited on 2017-04-27]. Available from: `http://www.cisco.com/en/US/docs/ios/solutions_docs/netflow/nfwhite.html`.
9. CLAISE, B. *Cisco Systems NetFlow Services Export Version 9* [RFC 3954 (Informational)]. Fremont, CA, USA: RFC Editor, 2004. Internet Request for Comments, no. 3954. ISSN 2070-1721. Available from DOI: `10.17487/RFC3954`.
10. CISCO SYSTEMS, INC., SAN JOSE, CA AND USA. *Cisco IOS NetFlow and Security* [online]. 2005 [visited on 2017-04-27]. Available from: `http://www.cisco.com/en/US/prod/collateral/iosswrel/ps6537/ps6586/ps6642/prod_presentation0900aecd80311f49.pdf`.
11. CISCO SYSTEMS, INC., SAN JOSE, CA AND USA. *Cisco IOS Flexible NetFlow* [online]. 2008 [visited on 2017-04-27]. Available from: `http://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/flexible-netflow/product_data_sheet0900aecd804b590b.html`.
12. THE INTERNET ENGINEERING STEERING GROUP. *IP Flow Information Export (ipfix) Charter* [online] [visited on 2017-04-27]. Available from: `http://datatracker.ietf.org/wg/ipfix/charter/`.
13. THE INTERNET ENGINEERING STEERING GROUP. *IP Flow Information Export Charter* [online]. 2001 [visited on 2017-04-27]. Available from: `https://www.ietf.org/mail-archive/web/ipfix/current/msg00213.html`.
14. QUITTEK, J.; ZSEBY, T.; CLAISE, B.; ZANDER, S. *Requirements for IP Flow Information Export (IPFIX)* [RFC 3917 (Informational)]. Fremont, CA, USA: RFC Editor, 2004. Internet Request for Comments, no. 3917. ISSN 2070-1721. Available from DOI: `10.17487/RFC3917`.
15. LEINEN, S. *Evaluation of Candidate Protocols for IP Flow Information Export (IPFIX)* [RFC 3955 (Informational)]. Fremont, CA, USA: RFC Editor, 2004. Internet Request for Comments, no. 3955. ISSN 2070-1721. Available from DOI: `10.17487/RFC3955`.
16. TRAMMELL, Brian; BOSCHI, Elisa. An Introduction to IP Flow Information Export (IPFIX). *IEEE Communications Magazine*. 2011, vol. 49, no. 4, pp. 89–95. ISSN 0163-6804. Available from DOI: `10.1109/MCOM.2011.5741152`.
17. TRAMMELL, B.; BOSCHI, E. *Bidirectional Flow Export Using IP Flow Information Export (IPFIX)* [RFC 5103 (Proposed Standard)]. Fremont, CA, USA: RFC Editor, 2008. Internet Request for Comments, no. 5103. ISSN 2070-1721. Available from DOI: `10.17487/RFC5103`.

18. SADASIVAN, G.; BROWNLEE, N.; CLAISE, B.; QUITTEK, J. *Architecture for IP Flow Information Export* [RFC 5470 (Informational)]. Fremont, CA, USA: RFC Editor, 2009. Internet Request for Comments, no. 5470. ISSN 2070-1721. Available from DOI: `10.17487/RFC5470`. Updated by RFC 6183.
19. BOSCHI, E.; MARK, L.; CLAISE, B. *Reducing Redundancy in IP Flow Information Export (IPFIX) and Packet Sampling (PSAMP) Reports* [RFC 5473 (Informational)]. Fremont, CA, USA: RFC Editor, 2009. Internet Request for Comments, no. 5473. ISSN 2070-1721. Available from DOI: `10.17487/RFC5473`.
20. DIETZ, T.; KOBAYASHI, A.; CLAISE, B.; MUENZ, G. *Definitions of Managed Objects for IP Flow Information Export* [RFC 5815 (Proposed Standard)]. Fremont, CA, USA: RFC Editor, 2010. Internet Request for Comments, no. 5815. ISSN 2070-1721. Available from DOI: `10.17487/RFC5815`. Obsoleted by RFC 6615.
21. DIETZ, T.; KOBAYASHI, A.; CLAISE, B.; MUENZ, G. *Definitions of Managed Objects for IP Flow Information Export* [RFC 6615 (Proposed Standard)]. Fremont, CA, USA: RFC Editor, 2012. Internet Request for Comments, no. 6615. ISSN 2070-1721. Available from DOI: `10.17487/RFC6615`.
22. AITKEN, P.; CLAISE, B.; S, S. B; MCDOWALL, C.; SCHOENWAELDER, J. *Exporting MIB Variables Using the IP Flow Information Export (IPFIX) Protocol* [RFC 8038 (Proposed Standard)]. Fremont, CA, USA: RFC Editor, 2017. Internet Request for Comments, no. 8038. ISSN 2070-1721. Available from DOI: `10.17487/RFC8038`.
23. KOBAYASHI, A.; CLAISE, B. *IP Flow Information Export (IPFIX) Mediation: Problem Statement* [RFC 5982 (Informational)]. Fremont, CA, USA: RFC Editor, 2010. Internet Request for Comments, no. 5982. ISSN 2070-1721. Available from DOI: `10.17487/RFC5982`.
24. KOBAYASHI, A.; CLAISE, B.; MUENZ, G.; ISHIBASHI, K. *IP Flow Information Export (IPFIX) Mediation: Framework* [RFC 6183 (Informational)]. Fremont, CA, USA: RFC Editor, 2011. Internet Request for Comments, no. 6183. ISSN 2070-1721. Available from DOI: `10.17487/RFC6183`.
25. BOSCHI, E.; TRAMMELL, B. *IP Flow Anonymization Support* [RFC 6235 (Experimental)]. Fremont, CA, USA: RFC Editor, 2011. Internet Request for Comments, no. 6235. ISSN 2070-1721. Available from DOI: `10.17487/RFC6235`.

26. MUENZ, G.; CLAISE, B.; AITKEN, P. *Configuration Data Model for the IP Flow Information Export (IPFIX) and Packet Sampling (PSAMP) Protocols* [RFC 6728 (Proposed Standard)]. Fremont, CA, USA: RFC Editor, 2012. Internet Request for Comments, no. 6728. ISSN 2070-1721. Available from DOI: `10.17487/RFC6728`.

27. CLAISE, B.; TRAMMELL, B.; AITKEN, P. *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information* [RFC 7011 (Internet Standard)]. Fremont, CA, USA: RFC Editor, 2013. Internet Request for Comments, no. 7011. ISSN 2070-1721. Available from DOI: `10.17487/RFC7011`.

28. BROWNLEE, Nevil. Flow-Based Measurement: IPFIX Development and Deployment. *IEICE Transactions on Communications*. 2011, vol. E94.B, no. 8, pp. 2190–2198. Available from DOI: `10.1587/transcom.E94.B.2190`.

29. PHAAL, Peter. *sFlow Version 5* [online]. 2004 [visited on 2017-04-27]. Available from: `http://sflow.org/sflow_version_5.txt`.

30. PHAAL, P.; PANCHEN, S.; MCKEE, N. *InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks* [RFC 3176 (Informational)]. Fremont, CA, USA: RFC Editor, 2001. Internet Request for Comments, no. 3176. ISSN 2070-1721. Available from DOI: `10.17487/RFC3176`.

31. INTERNET ENGINEERING TASK FORCE. *Packet Sampling (psamp) WG* [online] [visited on 2017-04-27]. Available from: `https://datatracker.ietf.org/wg/psamp/`.

32. THE INTERNET ENGINEERING STEERING GROUP. *Packet Sampling Charter* [online] [visited on 2017-04-27]. Available from: `https://datatracker.ietf.org/doc/charter-ietf-psamp/`.

33. DIETZ, T.; CLAISE, B.; AITKEN, P.; DRESSLER, F.; CARLE, G. *Information Model for Packet Sampling Exports* [RFC 5477 (Proposed Standard)]. Fremont, CA, USA: RFC Editor, 2009. Internet Request for Comments, no. 5477. ISSN 2070-1721. Available from DOI: `10.17487/RFC5477`.

34. ZSEBY, T.; MOLINA, M.; DUFFIELD, N.; NICCOLINI, S.; RASPALL, F. *Sampling and Filtering Techniques for IP Packet Selection* [RFC 5475 (Proposed Standard)]. Fremont, CA, USA: RFC Editor, 2009. Internet Request for Comments, no. 5475. ISSN 2070-1721. Available from DOI: `10.17487/RFC5475`.

35. CLAISE, B.; JOHNSON, A.; QUITTEK, J. *Packet Sampling (PSAMP) Protocol Specifications* [RFC 5476 (Proposed Standard)]. Fremont, CA, USA: RFC Editor, 2009. Internet Request for Comments, no. 5476. ISSN 2070-1721. Available from DOI: `10.17487/RFC5476`.

36. OPEN NETWORKING FOUNDATION. *OpenFlow Switch Specification* [online]. 2012 [visited on 2017-04-27]. Available from: `https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.1.pdf`.

37. SINGH, Sanjeev; JHA, Rakesh Kumar. A Survey on Software Defined Networking: Architecture for Next Generation Network. *J. Netw. Syst. Manage.* 2017, vol. 25, no. 2, pp. 321–374. ISSN 1064-7570. Available from DOI: `10.1007/s10922-016-9393-9`.

38. HU, Fei; HAO, Qi; BAO, Ke. A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation. *IEEE Communications Surveys Tutorials*. 2014, vol. 16, no. 4, pp. 2181–2206. ISSN 1553-877X. Available from DOI: `10.1109/COMST.2014.2326417`.

39. YU, Curtis; LUMEZANU, Cristian; ZHANG, Yueping; SINGH, Vishal; JIANG, Guofei; MADHYASTHA, Harsha V. FlowSense: Monitoring Network Utilization with Zero Measurement Cost. In: *Proceedings of the 14th International Conference on Passive and Active Measurement*. Hong Kong, China: Springer-Verlag, 2013, pp. 31–41. PAM'13. ISBN 978-3-642-36515-7. Available from DOI: `10.1007/978-3-642-36516-4_4`.

40. HOFSTEDE, Rick; DRAGO, Idilio; SPEROTTO, Anna; PRAS, Aiko. Flow Monitoring Experiences at the Ethernet-layer. In: *Proceedings of the 17th International Conference on Energy-aware Communications*. Dresden, Germany: Springer-Verlag, 2011, pp. 134–145. EUNICE'11. ISBN 978-3-642-23540-5. Available also from: `http://dl.acm.org/citation.cfm?id=2040416.2040437`.

41. GARCÍA-DORADO, José Luis; MATA, Felipe; RAMOS, Javier; SANTIAGO DEL RÍO, Pedro M.; MORENO, Victor; ARACIL, Javier. High-Performance Network Traffic Processing Systems Using Commodity Hardware. In: BIERSACK, Ernst; CALLEGARI, Christian; MATIJASEVIC, Maja (eds.). *Data Traffic Monitoring and Analysis*. Springer Berlin Heidelberg, 2013, vol. 7754, pp. 3–27. Lecture Notes in Computer Science. ISBN 978-3-642-36783-0. Available from DOI: `10.1007/978-3-642-36784-7_1`.

42. BUJLOW, Tomasz; CARELA-ESPANOL, Valentín; BARLET-ROS, Pere. Independent Comparison of Popular DPI Tools for Traffic Classification. *Computer Networks*. 2015, vol. 76, pp. 75–89. Available from DOI: `http://dx.doi.org/10.1016/j.comnet.2014.11.001`.

43. VELAN, Petr; ČERMÁK, Milan; ČELEDA, Pavel; DRAŠAR, Martin. A Survey of Methods for Encrypted Traffic Classification and Analysis. *International Journal of Network Management*. 2015, vol. 25, no. 5, pp. 355–374. Available from DOI: `10.1002/nem.1901`.

44. RODRIGUEZ, Juan Moliná; ESPAÑOL, Valentín Carela; ROS, Pere Barlet; HOFFMANN, Ralf; DEGNER, Klaus. Empirical Analysis of Traffic to Establish a Profiled Flow Termination Timeout. In: *2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC)*. 2013, pp. 1156–1161. ISSN 2376-6492. Available from DOI: `10.1109/IWCMC.2013.6583720`.

45. BRAUCKHOFF, Daniela; TELLENBACH, Bernhard; WAGNER, Arno; MAY, Martin; LAKHINA, Anukool. Impact of Packet Sampling on Anomaly Detection Metrics. In: *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*. Rio de Janeriro, Brazil: ACM, 2006, pp. 159–164. IMC '06. ISBN 1-59593-561-4. Available from DOI: `10.1145/1177080.1177101`.

46. VELAN, Petr; PUŠ, Viktor. High-Density Network Flow Monitoring. In: *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. 2015, pp. 996–1001. ISSN 1573-0077. Available from DOI: `10.1109/INM.2015.7140424`.

47. KEKELY, Lukas; KUCERA, Jan; PUS, Viktor; KORENEK, Jan; VASILAKOS, Athanasios V. Software Defined Monitoring of Application Protocols. *IEEE Trans. Comput.* 2016, vol. 65, no. 2, pp. 615–626. ISSN 0018-9340. Available from DOI: `10.1109/TC.2015.2423668`.

48. ALCOCK, Shane; NELSON, Richard. *Libprotoident: Traffic Classification Using Lightweight Packet Inspection* [Online article]. 2012 [visited on 2017-04-28]. Available from: `http://www.wand.net.nz/~salcock/lpi/lpi.pdf`. Technical report. WAND Network Research Group.

49. SADRE, Ramin; SPEROTTO, Anna; PRAS, Aiko. The Effects of DDoS Attacks on Flow Monitoring Applications. In: *2012 IEEE Network Operations and Management Symposium*. 2012, pp. 269–277. ISSN 1542-1201. Available from DOI: `10.1109/NOMS.2012.6211908`.

50. WANG, Dawei; XUE, Yibo; DONG, Yingfei. Memory-Efficient Hypercube Flow Table for Packet Processing on Multi-Cores. In: *2011 IEEE Global Telecommunications Conference - GLOBECOM 2011*. 2011, pp. 1–6. ISSN 1930-529X. Available from DOI: `10.1109/GLOCOM.2011.6134338`.

51. NASSOPULOS, Georges; ROSSI, Dario; GRINGOLI, Francesco; NAVA, Lorenzo; DUSI, Maurizio; SANTIAGO DEL RIO, Pedro Maria. Flow Management at Multi-Gbps: Tradeoffs and Lessons Learned. In: DAINOTTI, Alberto; MAHANTI, Anirban; UHLIG, Steve (eds.). *Traffic Monitoring and Analysis: 6th International Workshop, TMA 2014, London, UK, April 14, 2014. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 1–14. ISBN 978-3-642-54999-1. Available from DOI: `10.1007/978-3-642-54999-1_1`.

52. MOLINA, Maurizio.; CHIOSI, Agostino.; D'ANTONIO, Salvatore; VENTRE, Giorgio. Design Principles and Algorithms for Effective High-speed IP Flow Monitoring. *Comput. Commun.* 2006, vol. 29, no. 10, pp. 1653–1664. ISSN 0140-3664. Available from DOI: `10.1016/j.comcom.2005.07.024`.

A List of Authored Publications

A.1 Impacted Journals

1. VELAN, Petr; ČERMÁK, Milan; ČELEDA, Pavel; DRAŠAR, Martin. A Survey of Methods for Encrypted Traffic Classification and Analysis. *International Journal of Network Management*. 2015, vol. 25, no. 5, pp. 355–374. Available from DOI: `10.1002/nem.1901`

A.2 Conference Proceedings

1. VELAN, Petr; MEDKOVÁ, Jana; JIRSÍK, Tomáš; ČELEDA, P. Network Traffic Characterisation Using Flow-Based Statistics. In: *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*. 2016, pp. 907–912. Available from DOI: `10.1109/NOMS.2016.7502924`

2. VELAN, Petr. EventFlow: Network Flow Aggregation Based on User Actions. In: *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*. 2016, pp. 767–771. Available from DOI: `10.1109/NOMS.2016.7502895`

3. VELAN, Petr; PUŠ, Viktor. High-Density Network Flow Monitoring. In: *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. 2015, pp. 996–1001. ISSN 1573-0077. Available from DOI: `10.1109/INM.2015.7140424`

4. PUŠ, Viktor; VELAN, Petr; KEKELY, Lukáš; KOŘENEK, Jan; MINAŘÍK, Pavel. Hardware Accelerated Flow Measurement of 100 Gb Ethernet. In: BADONNEL, Remi; XIAO, Jin; ATA, Shingo; DE TURCK, Filip; GROZA, Voicu; SANTOS, Carlos Raniery P. dos (eds.). *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. Ottawa, Canada: IEEE Xplore Digital Library, 2015, pp. 1147–1148. ISSN 1573-0077. Available from DOI: `10.1109/INM.2015.7140452`

5. HUSÁK, Martin; VELAN, Petr; VYKOPAL, Jan. Security Monitoring of HTTP Traffic Using Extended Flows. In: *2015 10th International Conference on Availability, Reliability and Security*. 2015, pp. 258–265. Available from DOI: `10.1109/ARES.2015.42`

6. VELAN, Petr; ČELEDA, Pavel. Next Generation Application-Aware Flow Monitoring. In: SPEROTTO, Anna; DOYEN, Guillaume; LATRÉ,

Steven; CHARALAMBIDES, Marinos; STILLER, Burkhard (eds.). *Monitoring and Securing Virtualized Networks and Services*. Springer Berlin Heidelberg, 2014, vol. 8508, pp. 173–178. Lecture Notes in Computer Science. ISBN 978-3-662-43861-9. ISSN 0302-9743. Available from DOI: `10.1007/978-3-662-43862-6_20`

7. VELAN, Petr; JIRSÍK, Tomáš; ČELEDA, Pavel. Design and Evaluation of HTTP Protocol Parsers for IPFIX Measurement. In: BAUSCHERT, Thomas (ed.). *Advances in Communication Networking*. Heidelberg: Springer Berlin Heidelberg, 2013, vol. 8115, pp. 136–147. ISBN 978-3-642-40551-8. Available from DOI: `10.1007/978-3-642-40552-5_13`

8. VELAN, Petr. Practical Experience with IPFIX Flow Collectors. In: DE TURCK, Filip; DIAO, Yixin; SE ON HONG, Choong; MEDHI, Deep; SADRE, Ramin (eds.). *IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*. Ghent, Belgium: IEEE Xplore Digital Library, 2013, pp. 1021–1026. ISBN 978-1-4673-5229-1. Available also from: `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6573125`

9. ELICH, Martin; VELAN, Petr; JIRSÍK, Tomáš; ČELEDA, Pavel. An Investigation Into Teredo and 6to4 Transition Mechanisms: Traffic Analysis. In: DAMLA TURGUT Nils Aschenbruck, Jens Tölle (ed.). *IEEE 38th Conference on Local Computer Networks Workshops (LCN Workshops)*. Sydney, Australia: IEEE Xplore Digital Library, 2013, pp. 1046–1052. ISBN 978-1-4799-0540-9. Available from DOI: `10.1109/LCNW.2013.6758546`

10. ČELEDA, Pavel; VELAN, Petr; RÁBEK, Martin; HOFSTEDE, Rick; PRAS, Aiko. Large-Scale Geolocation for NetFlow. In: DE TURCK, Filip; DIAO, Yixin; SE ON HONG, Choong; MEDHI, Deep; SADRE, Ramin (eds.). *IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*. Ghent, Belgium: IEEE Xplore Digital Library, 2013, pp. 1015–1020. ISBN 978-1-4673-5229-1. Available also from: `http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6573124`

11. VELAN, Petr; KREJČÍ, Radek. Flow Information Storage Assessment Using IPFIXcol. In: SADRE, Ramin; NOVOTNÝ, Jiří; ČELEDA, Pavel; WALDBURGER, Martin; STILLER, Burkhard (eds.). *Dependable Networks and Services*. Heidelberg: Springer Berlin Heidelberg, 2012, vol. 7279, pp. 155–158. Lecture Notes in Computer Science. ISBN 978-3-642-30632-7. Available from DOI: `10.1007/978-3-642-30633-4_21`

A.3 Other Publications