Masaryk University
Faculty of Informatics

Application-Aware Flow Monitoring

Doctoral Thesis

Petr Velan

Brno, Spring 2018

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

I would like to thank my advisor, colleagues, and fellow researchers for their input and insights that helped to shape my research and this thesis. Furthermore, I could have hardly finished this work without the relentless support of my friends and family. They have my eternal gratitude.

Abstract

The Internet has become a crucial medium for business, entertainment, communication, learning, access to information, and other human endeavours. As the society becomes increasingly dependent upon the availability and security of online services, the revenue from disrupting the normal operation of these services increases as well. Gaining unauthorised access, disrupting functionality, stealing and selling confidential data, and impersonation are only a few examples of these disruptions. Various systems, such as firewalls and anti-virus software, are intended to thwart these attacks and reduce the associated risks. To protect users on their networks, administrators often deploy network monitoring systems to detect and mitigate the threats.

Network flow monitoring is an instrument which allows observing traffic passing through given point in the network and gathering aggregated information about the observed network connections. The aggregated nature of provided information allows to scale the flow monitoring to high-speed networks and monitor traffic rates up to and including hundreds of gigabits per second. Flow data is mainly used for security purposes. However, due to its versatility, it is often used for network management tasks, such as capacity planning and accounting, as well. As both network communication protocols and attack vectors are becoming increasingly sophisticated, the flow monitoring needs to evolve as well. The current trends for flow monitoring are the extraction of supplementary information from application protocols, identification of encrypted traffic, and monitoring of high-speed networks.

This thesis contributes to the progression of flow monitoring by exploring the possibilities unlocked by extending the flow data with application-specific information. We show how the construction of flows is affected by the addition, present the benefits to traffic analysis and assess the inevitable performance loss. To compensate for the lost performance several novel optimisation techniques are proposed for the flow monitoring process. Recognizing that the increasing deployment of encryption is going to limit the benefits of application flow monitoring, we perform a survey of methods for measurement of encrypted traffic. The thesis is concluded by an outlook towards future possibilities for flow monitoring advancement.

The first contribution of this thesis is a revised definition of flow that attempts to improve and update currently used definitions so that it better matches current flow monitoring practices. A formal definition of flow, together with algorithms for flow construction based on this definition is provided as well. We demonstrate that the revised flow definition allows for use cases that were not covered by NetFlow v9 and IPFIX definitions, such as monitoring of traffic containing fragmented packets.

Using the revised terminology and definitions, we focus on application flow monitoring, which is one of the current trends in flow monitoring. Firstly, an overview of the state of application flow monitoring is provided. Secondly, practical definitions of application flow and application flow record are given to circumscribe application flow monitoring. Thirdly, an experimental study on the design of an HTTP application protocol parser is presented. The study quantifies how application flow monitoring increases the demand for computational resources and decreases the performance of the flow monitoring system.

The application flow monitoring provides new data for traffic analysis. We study several use cases for which the application flow monitoring can be applied in detail. The first one uses information from HTTP headers to detect new classes of attacks on the application layer. The second one shows how the additional information can be used to analyse utilisation of IPv6 transition mechanisms. Then, we show that adding geolocation information to flow records can be used for advanced traffic analysis. And last, a method for characterising network traffic is described. It allows comparing multiple network traces and searching for similarities.

The monitoring of high-speed networks is one of the current trends in flow monitoring. Firstly, we describe the state-of-the-art in this area. Secondly, we identify and propose multi-

ple optimisations including those that rely on programmable network interface cards. Some of these optimisations are then used to build and demonstrate a high-density flow monitoring system capable of processing sixteen 10 Gb links in a single box.

Classification of encrypted traffic and identification of applications using encryption has become a widely researched topic in the last decade. A survey of methods for measurement of encrypted traffic is, therefore, one of the contributions of this thesis. We show that surprisingly detailed information can be obtained using these methods. In specific cases, even the content of the encrypted connection can be established.

This work concludes with a vision for the future of the flow monitoring. We identify several directions of future research of flow monitoring and a novel approach to monitoring of tunnelled traffic and application layer is proposed. We believe that the perception of the whole application flow monitoring should be revised to facilitate future demands for more complete and better-structured data.

Keywords

network, monitoring, measurement, flow, application flow, NetFlow, IPFIX, encryption, performance, 100 Gbps

Contents

Chapter 1

Introduction

The concept of network flow monitoring was proposed in 1991 to facilitate accounting of network usage. Cisco's NetFlow became a de-facto standard for acquiring IP network and operational data in the following years. However, the main potential of flow monitoring became realised much later, in 2005, when Cisco engineers proposed to use NetFlow for anomaly detection and traffic analysis [1]. Flows records together with packet capture are the two main sources of data for intrusion detection systems nowadays. Moreover, the flow records are being used for data retention [2], which is mandatory for internet service providers in many countries.

Due to the growing cybercrime industry and cyber espionage [3] the traffic analysis and security potential of flow monitoring have become more accentuated over time. To support traffic analysis, Cisco enriched its Flexible NetFlow [4] with information from the Network-Based Application Recognition (NBAR) [5] in 2009. NBAR was initially used for QoS management on Cisco appliances. The trend of enriching flow records with information from application layer continued and resulted in the application-aware flow monitoring, which can be seen as a combination of Deep Packet Inspection (DPI) and flow monitoring. A deployment of flow monitoring has become standard practice since almost every enterprise networking equipment is able to export flow records nowadays. Steinberger et al. surveyed ISPs and network operators and found out that a majority of them uses flow monitoring for attack detection in their networks [6].

The importance of flow monitoring is growing. As the speed of network links increases, DPI-based intrusion detection systems are becoming incapable of handling the sheer amount of traffic. Moreover, increasing amount of encryption makes it harder still for DPI to be effective. The goal of this thesis is to advance flow monitoring techniques to achieve better application visibility and performance.

1.1 Problem Statement

When the flow monitoring is deployed on a network, the measured flow data are sent to a flow data processing system. The system facilitates flow analysis, reporting, and threat detection. All of these functions require high-quality flow data for their operation. For example, when some of the packets are not monitored or actively sampled, the quality of flow data is significantly reduced [7]. Moreover, when the data contains artifacts [8], the data analysis and threat detection can be impaired as well.

Maintaining high-quality flow monitoring system is a challenging task due to the constant changes in traffic structure and increasing volume [9]. We have identified three main topics that directly affect flow monitoring and flow data analysis. Firstly, the flow monitoring must keep pace with the increasing speed of networks. Therefore, the performance of flow monitoring must be studied and improved to match the speed of the network links. Secondly, as the network attacks are becoming more sophisticated, application layer information must be provided to

enable more efficient threat detection. Lastly, the increasing amount of encrypted traffic makes application visibility difficult. Therefore, to maintain any degree of application visibility, novel approaches for monitoring of encrypted traffic must be explored.

1.1.1 Application Layer Information

When we started our research in flow monitoring in 2012, application visibility in flow monitoring was a relatively new concept. With the exception of Flexible NetFlow utilising NBAR, the flow records contained only network and transport layer information. However, even the Flexible NetFlow provided only application recognition without extraction of any application-specific data. At the end of 2011 ntop released a version of their nProbe flow exporter capable of utilising OpenDPI library to provide information about application protocol [10]. Still, the information provided was only application protocol name and identifier, which was very similar to what the Flexible NetFlow provided.

With the increasing number of discovered vulnerabilities in applications [11], we have found it essential to increase the capabilities of flow monitoring to detect more of these vulnerabilities. Therefore, we have decided to create true application-aware flow monitoring that would allow threat detection algorithms to utilise not only network and transport layer information, but also application layer information as well.

1.1.2 Growing Network Speeds

The implementation of flow monitoring started as an additional feature of routers and switches. However, as the main purpose of the networking devices is not flow monitoring, the quality of data is compromised under high load, where the networking capabilities are of higher importance than the monitoring itself. This, and the reason that the devices provided only limited configuration options lead to the development of flow monitoring on commodity hardware [12]. It was shown that even monitoring of a Gigabit Ethernet on commodity hardware is a challenging problem. Therefore, even with increasing CPU frequency and the number of cores, flow monitoring of 10 Gbps and faster networks requires the use of special techniques and optimisations. The recent advances in networking technologies lead to standardisation and deployment of 40 Gbps and 100 Gbps Ethernet links. Moreover, standards for 200 Gpbs and 400 Gbps Ethernet were approved at the end of 2017.

We have decided to research the possibilities of high-speed flow monitoring at these speeds. Moreover, since application-aware flow monitoring requires more performance than basic flow monitoring, we study the impact of processing of application payloads on the flow monitoring performance.

1.1.3 Traffic Encryption

Our decision to include application layer information in flows to increase network visibility and aid threat detection was based partially on the amount of unencrypted traffic that could be observed in the network. Research showed that only one-third of web pages could be browsed via HTTPS [13] in 2013. However, the amount of encrypted traffic steadily increased, and more than 70 percent of web pages are loaded over HTTPS nowadays [14]. This massive change in the use of encryption necessitates the use of novel approaches to monitoring of the encrypted traffic. Therefore, a study of encrypted protocols, as well as an overview statistical methods of traffic classification, are needed to analyse the impact of encryption on the amount of information that can be provided by the flow monitoring.

1.2 Research Goals

The goals of this thesis are motivated by the discovered problems. We have identified the following list of goals:

- Propose application flow monitoring which utilises application layer information to facilitate flow analysis and threat detection.
- Evaluate performance of flow monitoring and propose optimisations to facilitate monitoring of high-speed networks.
- Analyse options for monitoring of encrypted traffic, survey common encryption protocols and methods for encrypted traffic classification.

1.3 Contributions

In the course of pursuing the research goals, following contributions were made in the area of flow monitoring:

- We have proposed a new definition of flow which respects the nature of flow monitoring. A formalisation of the flow definition is provided to make the definition more expressive. Moreover, we have shown how the definition can be used to describe the flow creation process formally as well. (Chapter 2)
- We provide an overview of application flow monitoring that has been implemented in the past years. Consistent terminology is lacking in the area of application flow monitoring, which leads us to propose a definition of application flow monitoring together with appropriate terminology. (Chapter 3)
- HTTP application flow monitoring has been implemented, and its performance evaluated. We show that evaluation of application monitoring performance depends on the used dataset even more heavily than basic flow monitoring. (Chapter 3)
- The effect of additional information provided by application flow monitoring has been investigated on several different protocols. We have also analysed the feasibility and benefits of adding geolocation information to flow. (Chapter 4)
- Flow monitoring performance of commodity hardware was analysed, and multiple optimisations were proposed. The use of hardware acceleration using FPGA-based network interface cards and possible optimisations with use of these cards were discussed as well. A monitoring system theoretically capable of achieving 160 Gbps throughput was built based on the proposed optimisations. (Chapter 5)
- Analysis of the most widely used encryption protocols was performed, and information suitable for use in application flow monitoring was discovered in the headers of these protocols. Traffic classification methods based on various statistical methods and machine learning were analysed. However, the usefulness of these methods for flow monitoring remains for further research. (Chapter 6)
- Three novel concepts that can benefit flow monitoring and especially application flow monitoring were presented. (Chapter 7)

1.4 Thesis Structure

The rest of this thesis is structured as follows. Chapter 2 describes flow monitoring, its history and present state as well as lessons learned from the deployment of flow monitoring system at CESNET National Research and Education Network. Chapter 3 discusses application flow monitoring, its definition and terminology. It also studies the implementation of application flow monitoring for the HTTP protocol and evaluates its performance. Chapter 4 consists of four use cases that highlight the benefits of application flow monitoring. Chapter 5 examines the performance of flow monitoring system on commodity hardware and a high-density monitoring system for high-speed networks is built and thoroughly evaluated. Chapter 6 analyses possible approaches to coping with monitoring of encrypted traffic. Chapter 7 proposes three novel concepts for application flow monitoring. Finally, Chapter 8 concludes the thesis and presents directions for future work.

Chapter 2

Network Flow Monitoring

Flow monitoring is an essential part of network accounting and security nowadays. It facilitates large-scale intrusion detection and prevention systems, data analysis, capacity planning, data retention, and other operations necessary for network management. The aim of this chapter is to provide a comprehensive introduction to network flow monitoring. Firstly, the history of flow monitoring is outlined, and related technologies are described and compared to flow monitoring. Secondly, the applicability of currently used flow definition is discussed, and an improved definition is proposed, together with a formal notation. The formal notation allows clear description of flow related algorithms and avoids misunderstandings caused by ambiguities of a natural language. To the best of our knowledge, this is the first attempt to formalise the flow creation process that takes into account all input data used in practice. The rest of the chapter presents the flow monitoring process in detail. Issues encountered while deploying and operating flow monitoring infrastructure of CESNET National Research and Education Network are discussed at the end of this chapter.

With the exception of the novel flow definition, this chapter covers a similar topic as the article of Hofstede et al. [15]. The article served as one of the sources of information about the history of flow monitoring and as an inspiration for the structure of this chapter. The reader is encouraged to study the article as an additional source of information. It focuses more on the IPFIX protocol and contains extensive information about the capabilities of existing enterprise and open-source software for flow creation and processing.

The paper included in this chapter is [A1].

The organisation of this chapter is as follows:

- *Section 2.1 introduces flow monitoring and provides overview of its history and related technologies.*
- *Section 2.2 provides an improved flow definition and based on this definition provides formal description of flow creation process.*
- *Section 2.3 explains flow monitoring architecture.*
- *Section 2.4 describes the flow monitoring process from packet capture to flow record export.*
- *Section 2.5 describes flow data treatment from flow record reception to storage and further processing.*
- *Section 2.6 discusses common issues encountered in flow monitoring operation.*
- *Section 2.7 summarizes the chapter.*

2.1 Flow Monitoring Basics

This section describes the history of flow monitoring, its development, and standardisation efforts. Technologies related to flow monitoring are discussed, and the principal differences are explained.

2.1.1 History of Flow Monitoring

The first mention of a flow export can be found in RFC 1272 [16] 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 utilisation. 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 [16]*

The goal at the time was to provide a framework for traffic accounting. However, the common belief 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 [17].

In 1995, Claffy et al. showed a methodology for internet traffic flow profiling based on packet aggregation [18], which started a revival of flow monitoring efforts. The Realtime Traffic Flow Measurement (RTFM) Working Group was created in 1996, and it had three primary objectives. The first was to consider current issues relating to traffic measurement, such as security, privacy, policies, and requirements on new network protocols. The 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 standard track IETF documents. The effort resulted in 1999 by publishing several RFCs describing new traffic flow measurement framework with increased flexibility and even provided bi-directional flow support [19]. 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 realised 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 flow export technology called NetFlow, it achieved widespread adoption. The main reason for the wide adoption was the fact that it was readily available on most Cisco devices with little effort. The NetFlow was patented in 1996 and the first version that became available to the general public around 2002 was NetFlow v5 [20], 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 the IPv6 protocol.

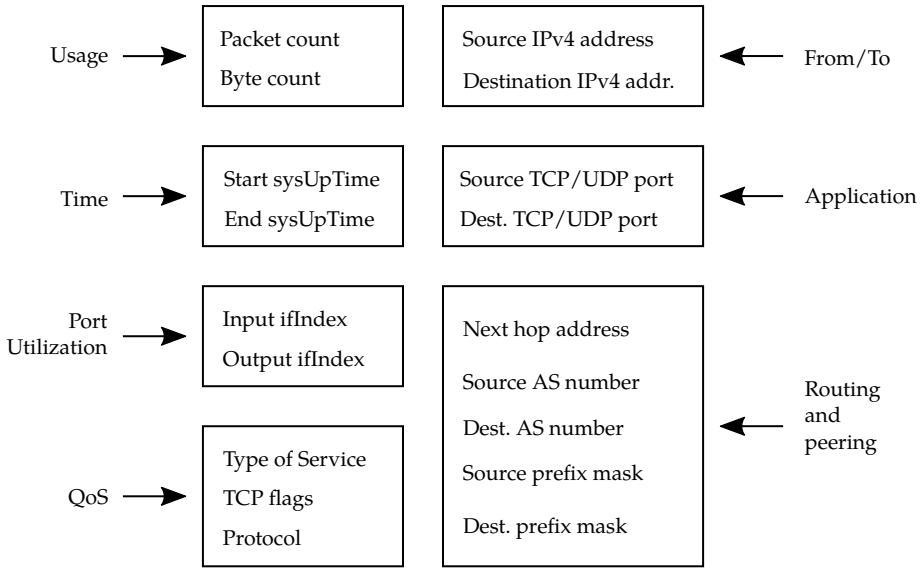


Figure 2.1: Fields exported by NetFlow v5 [20].

NetFlow v5 was soon obsoleted by NetFlow v9 which remedied some of the deficiencies of the previous version. The state of NetFlow v9 is described in [21]. It allowed defining an 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 protocols. When the potential of flow monitoring for security purposes became realised in 2005 [1], more effort was devoted to extending flow records with information not directly associated with switching. Cisco presented Flexible NetFlow technology [4] 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 was 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 [22]. The original charter [23] defined six specific goals for the WG:

- Define *"standard IP flow"*.
- Devise flow data encoding that supports 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.

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 a set of requirements for the IPFIX protocol [24] and evaluated existing

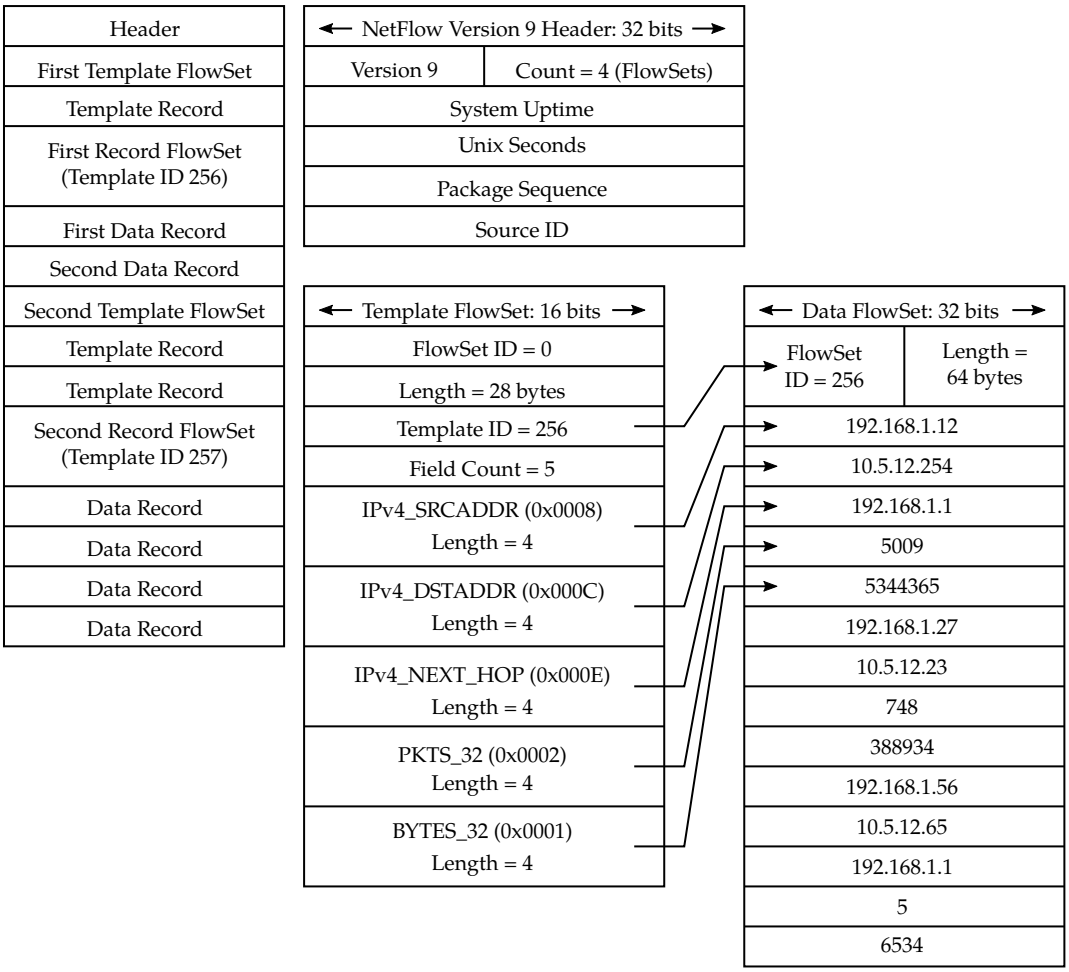


Figure 2.2: NetFlow v9 protocol structure example [20].

candidate protocols [25] to decide the most suitable approach to defining the new protocol. The NetFlow v9 specification (RFC 3954) was designed with IPFIX requirements in mind [26] 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, 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 published before its conclusion, the WG paid attention to, for example:

- Bidirectional flow export [27]
- Architecture for IP flow information export [28]
- Reducing redundancy in flow [29]
- Definitions of Managed Objects (MIB) for IPFIX [30, 31, 32]
- IP flow mediation framework [33, 34]
- IP flow anonymization [35]
- IPFIX configuration data model [36]

The IPFIX protocol specification is described by *"Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information"* [37] 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 [38].

2.1.2 Related Technologies

Flow monitoring is not the only network monitoring system used to gain information about network behaviour. There are other technologies that can be used to monitor network traffic and that can sometimes be confused with flow monitoring. We describe sFlow [39], IETF Packet Sampling, OpenFlow, and Deep Packet Inspection in the following text.

sFlow is an industry standard that is supported by a number of vendors in their packet switching devices. Its initial specification was published as an Informational RFC [40] 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 packets 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 entirely unrelated to flow monitoring. sFlow version 5 is the latest version and was published in 2004 [39].

In 2002 the IETF started Packet Sampling (PSAMP) Working Group [41] which was chartered to define a standard set of capabilities for network elements to sample subsets of packets by statistical and other methods [42]. The result is similar to sFlow; however, the PSAMP uses IPFIX protocol for data export [43]. The WG was concluded in 2009 after publishing four RFCs. The proposed standards include sampling and filtering techniques for IP packet selection [44], packet sampling protocol specifications [45], and information model for packet sampling export [43].

OpenFlow [46] is considered to be one of the first Software Defined Networking (SDN) standards [47, 48]. 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. [49] 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-SDN networking devices, there are significant differences. The control traffic itself is utilised to transfer data about new and expired flow records. Therefore, the configuration of flow monitoring is directly affected by the configuration of SDN network and vice versa. This imposes undesirable restrictions on the flow monitoring process. Hendriks et al. assess the quality of flow data from OpenFlow devices in [50] and report numerous problems with the measurement and resulting data quality. Suárez-Varela and Barlet-Ros propose a more scalable solution to mitigate some of the identified problems in [51]. However, the distributed architecture of the monitoring is usually 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 any 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 higher resources than standard flow monitoring, it provides maximum information about network traffic. DPI is an approach, rather than a 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 with the 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 is 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 describe the flow monitoring process accurately, we need to have a precise definition of what the flow is. The NetFlow v9 description in [21] 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 [21]*

The Observation Point is defined as a location where IP packets can be observed. According to the definition, a flow is a set of packets within a particular 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). Since this definition is quite generic, it covers most of the conventional IP flow creation techniques.

The IPFIX Protocol is an internet standard [37] 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 is 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 [37]*

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 are considered as part of the packet's characteristics. The total packet length can also be included here (it was even used as an example in the early drafts in 2002).

3. The IPFIX standard limits the definition of flows only to IP traffic. However, flows are often created with the use of link layer headers. Moreover, the flow concept works even for non-IP connections, e.g., in technological networks. Therefore, the generic flow definition should allow even non-IP packets. It should be noted that the NetFlow v9 definition of flow explicitly defines IP flows, not generic 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. Both NetFlow v9 and IPFIX define a set of common properties used to decide which flow the packet belongs to. This must be derived only from the single packet, which is not possible in case of fragmented packets.

In order to provide the 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 clearer about deriving data from previous packets of the same flow which is used for correctly handling 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 key* and *flow record*. 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
A flow key *is a set of common properties that is used to specify a* flow*.*

A flow record is basically a tuple containing the flow key and other properties measured for the flow. Moreover, we allow inclusion of more information about the flow that is derived from external sources. An example can be a name of a user to which particular IP address belonged at the time of measurement. The following definition reflects that:

Definition 2.3
A flow record *is a tuple which describes a particular* flow *containing values of:*

1. *the* flow key *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*,*
   (c) *external source of information.*

To make the definitions above clearer, we provide an example of real properties that might be contained in a flow record in Table 2.1. The table 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 the number of packets are involved. A summation function is usually applied to the number of bytes in each packet, TCP flags are aggregated using a logical OR function, the flow start timestamp is derived using a minimum function on each packet timestamp. The non-aggregated properties may be used as part of a flow key.

The definition of flow record states that each flow record describes a particular flow. Moreover, in the rest of this thesis, we make the assumption that each flow is described by a single flow record. This is particularly important for long-lived flows that are terminated by an active timeout. Either the whole flow can be terminated and a new one started, or the flow can continue and only the matching flow record can be expired. Multiple flow records are created in the latter case. We use the former interpretation since it allows to simplify the following definitions.

	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.

Definition 2.1 states what the flow is. Although we tried to be as explicit as possible, the definition uses informal language and is 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 the 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 $\widehat{p} \in \widehat{P}$ 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*:*

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

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

We denote a 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

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

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 that a packet belong to a flow if it 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 packet indexes 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 $\varphi((\widehat{p}_n)_{n\in\mathbb{I}}, \widehat{p}_k) = true$ 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 shall 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 $(p_i)_{i\in S}, (\widehat{p}_i)_{i\in S}, S \subseteq \mathbb{S}$ be 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*

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

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

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 an empty set (the cases where no flow is defined and where we have already added all possible indexes from $\mathbb{S}$), we define

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

Definition 2.6 of flow creates a 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 a 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.

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

1: Denote $S_1 = \mathbb{S}$
2: Set counter $i = 1$
3: **repeat**
4: Apply the *flow selection function* φ 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** $\mathcal{S}_i$ is empty

Algorithm 2.2: Construction of a Sequence of Flows.

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:*

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

Definition 2.7 provides a guide to constructing a sequence of flows. 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. In our definition, we want every packet to be a part of only a single flow. Therefore, we apply the *flow selection function* φ to each pair of existing flow (enriched by packet treatment information) and the new extended packet. Then, we add the packet to the first flow that matches. If none of the existing flows matches, we apply the function φ to this packet only and start a new flow if necessary.

From this, we can see that the flow creation process depends solely upon the implementation of the *flow selection function*. We will shortly discuss common implementations in the Subsection 2.4.3.

The condition for every packet to be part of exactly one flow might be too constricting in some cases. It is possible to remove the condition and simply start building each flow from a next extended packet in the sequence. However, this will create many overlapping flows that contain mostly the same packets but start with a different one. This problem would need to be addressed should such a definition be used.

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 processed 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 [34] 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.

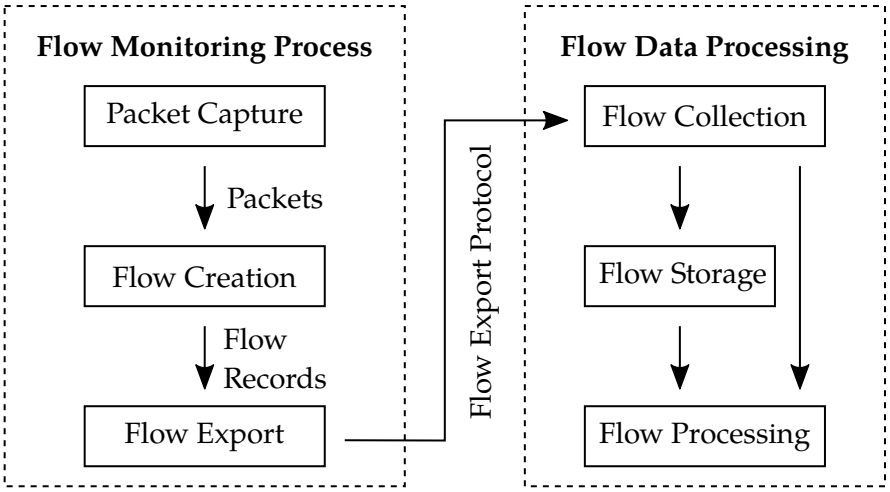


Figure 2.3: High Level Flow Monitoring Schema.

2.3.1 Terminology

The IPFIX working group published several documents where the architecture for IP Flow Information Export is described [28, 34]. 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 [28], 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.

We provide the IPFIX terminology definitions from RFC 7011 [37] here for the convenience of the reader:

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 [37]

The Figure 2.4 shows various scenarios of flow monitoring architecture as defined by IPFIX working group. IPFIX exporters and IPFIX devices are part of the flow monitoring process, collectors and application represent flow data processing, as described in the Figure 2.3. The IPFIX Reference Model [28] allows differentiating between devices that only export flow records and devices that receive data from an observation point and perform the metering process themselves. This can be useful for describing for example development setups where flow records are replayed or generated without the necessity for monitoring of live network traffic. Collectors comprise of collecting processes and data processing applications. The reference model shows that it is possible to collect flow records from multiple sources on a single collector and that the applications can run directly on the collector or that they can be distributed to other machines. For example, collecting process might convert the flow records in IPFIX format to JSON and feed a big data processing framework that runs on a cluster of machines.

Although the IPFIX Reference Model describes the flow monitoring architecture in detail, it is not used by the flow monitoring community unanimously. Most of the terms used in the IPFIX standard are simplified, and their true meaning often depends on the context. Table 2.2 provides the common terms often used instead of the IPFIX terminology. Examples of the use of the common terminology can be found for example in [15, 52, 38, 53, 54, 55, 56]. We usually talk about a monitored link and a set of monitored links (e.g., both directions of a network connection when optical fibres are used) instead of an observation point or an observation domain. By an exporter or a flow exporter is usually meant the software that performs flow monitoring (both metering and exporting process). If a device generates flows without observing and processing the packets first (i.e., Exporter in IPFIX terminology), we call it a flow source or a probe.

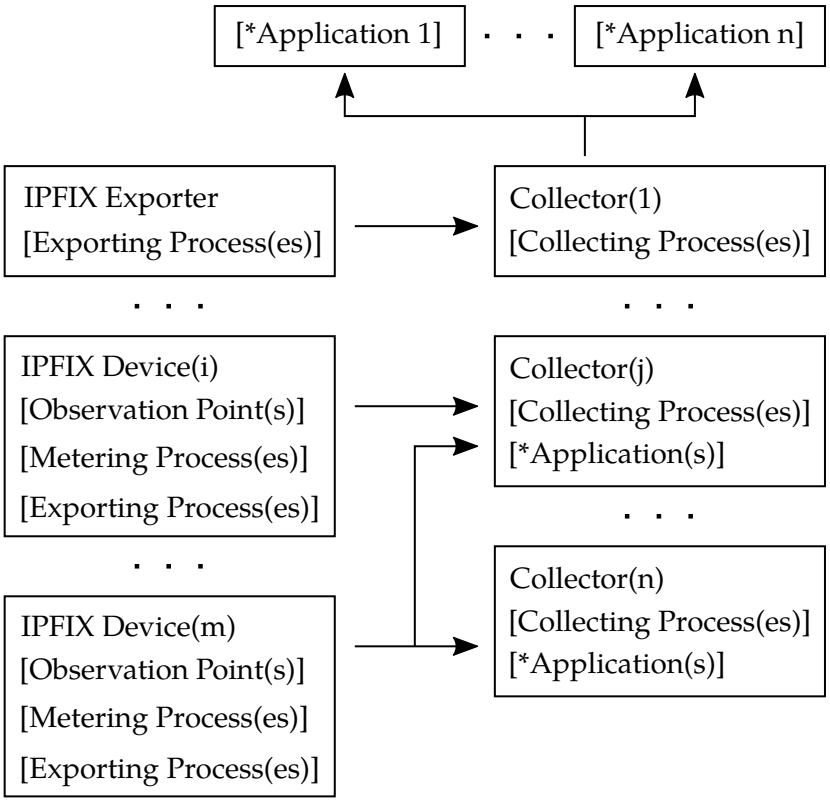


Figure 2.4: IPFIX Reference Model [28].

IPFIX Terminology	**Common Terminology**
Observation Point	Monitored Link, Observed Link
Observation Domain	Set of Monitored Links
Packet Treatment	Packet Treatment
Metering Process	(Flow) Exporter [software]
Exporting Process	(Flow) Exporter [software]
Exporter	Flow Source, (Flow) Probe
IPFIX Device	Flow Source, (Flow) Probe, Switch, Router, ...
Collecting Process	(Flow) Collector [software]
Collector	(Flow) Collector [device]

Table 2.2: Flow Monitoring Architecture Terminology.

The IPFIX Device is usually called by more specific name, such as a switch, router, or, in case of a dedicated device possibly with specialised hardware and software equipment, a probe. Furthermore, the term flow source can also be used for any device that generates flow records. Finally, both the software for collecting flow records and device where such software runs is commonly called a collector, or a flow collector. We will be using the common terminology throughout this thesis.

2.3.2 Flow Monitoring Deployment

The deployment of flow monitoring requires careful planning so that it does not disturb the existing network infrastructure. There are several decisions that must be made such as choosing a proper flow source, collector, and their location relative to the monitored network. Although

it is possible to monitor wireless and virtual networks, we focus on the deployment in wired networks.

The selection of the flow source depends upon many variables, such as cost, required quality of exported data, or the type of the monitored link. Two types of flow source are generally available. First are the active networking devices that are already present in the network and provide flow monitoring functionality as well. Switches, routers, and firewalls belong to this category. When such a device is present at a convenient point in the network, it is sufficient to configure it for flow export, and no adjustments to the network are needed. Moreover, internal information such as IP addresses behind NAT (Network Address Translation), which would be difficult to access otherwise, can be added to exported flow records. The disadvantage of these devices is that flow monitoring is not their primary function and may not be performed correctly under high load (i.e., under certain types of attack). Also, the range of supported options and protocols is usually much lower than that of dedicated probes. The reason for deploying a dedicated probe is usually the need for some extra functionality or guarantees that cannot be met by the networking devices. Furthermore, it allows separating network configuration and maintenance from network monitoring, which can be useful if they are handled by different divisions of an organisation.

The active networking devices observe packets as a part of their function, dedicated probes, however, need to be provided with access to data. There are two ways for probes to observe the data: in *in-line* mode or *mirrored* mode.

In-line mode – A device in in-line mode is connected directly to the monitored link and has to actively pass the packets in order for the link to function. This is the mode of operation of active network devices such as switches and routers. The advantage is that no other device is necessary to mirror the traffic, however, if the probe fails the operation of the network is disrupted. Moreover, it could introduce significant latency and jitter to the network connection.

Mirrored mode – In the mirrored mode, a copy of network traffic is created and delivered to the probe using a dedicated link. The copy can be created either by dedicated TAP (Test Access Point) or an active networking device with the use of a SPAN (Switch Port Analyzer) port. Table 2.3 shows differences, advantages, and disadvantages of TAP and SPAN solutions. An analysis of traffic trace artefacts caused by port mirroring was performed by Zhang and Moore in [57].

TAP is a passive splitting mechanism which requires in-line installation. However, due to the simplicity of the device (optical TAPs use mirrors and require no power source) and integrated fail-safes (bypass TAPs), the risk of negatively affecting the network is very low.

SPAN is a special port provided by active networking device that provides a copy of traffic passing through the device for analysis and monitoring purposes.

Since flow monitoring is a passive form of monitoring in its nature, it is a good practice to mirror the live traffic and provide only a copy of the data to the probe so that the network cannot be affected by the monitoring process. Selection of the mirroring technology needs to be considered for each deployment based on specific requirements and limitations, such as utilisation of the measured network and impact of packet drop on the performed analysis. The Figure 2.5 shows the most common deployment of flow monitoring at the edge of the network using both presented options: flow export directly from the router and dedicated probe with data provided either by TAP or by router SPAN port.

Location of the flow source in the network is essential. When the monitored network is connected to the outside at multiple points, it is necessary not only to monitor each of the points but

TAP	SPAN
Differences	
• RX & TX signal delivered on separate ports • Captures everything on the wire, including MAC and media errors • Complete capture for 100 % saturated network	• RX & TX copied into in one TX signal • Hardware and media errors are dropped • Possible packet drop due to SPAN link capacity limit
Advantages	
• Monitoring device receives identical data, including errors • Keeps link directions separate	• Low cost • No changes to network topology • Aggregation of multiple links
Disadvantages	
• Analysis device may need dual-port capture interface • Additional costs for TAP • Necessity to install additional device	• Packet drop on fully utilized full-duplex links • SPAN port data has lower priority than port-to-port data • Some analyses require observation of physical layer errors • Loss of information about link • Increased switch CPU utilization • Can change a timing of frames

Table 2.3: Differences Between TAP and SPAN Mirroring Options.

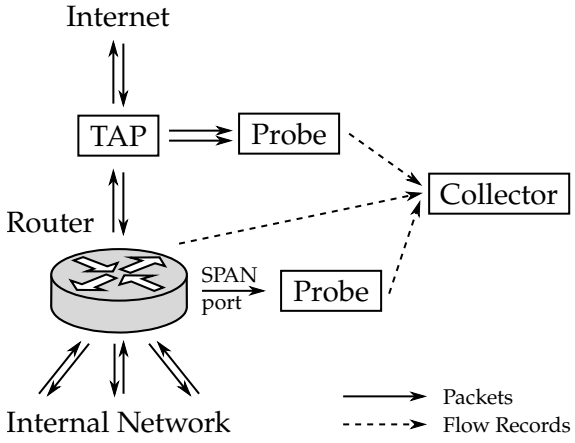


Figure 2.5: Flow Monitoring Deployment Schema.

also to be sure that the routing policies are reasonable, so that packets of each flow traverse only one of these points. The flow source would have to have access to all links and monitor them as a single source otherwise, which is not easily achieved, especially when the connection points are geologically distributed. It is a good practice to keep the information about which flow source exported which flow records on the collector in order to detect possible configuration problems more efficiently.

When monitoring a large network such as a network of a university campus, multiple flow sources can be utilised to gain more detailed information about traffic of individual faculties. Special care needs to be taken when a packet traverses multiple observation points. Counting the same traffic multiple times can have a negative impact on subsequent data analysis.

Deployment of NAT impedes network visibility. Some routers that perform the address translation are able to export both original and translated addresses in the flow records. However, observation points of dedicated flow probes are located either before or after the network device which performs the translation. Although some effort has been dedicated to detection of NAT from data provided in flow records [58, 53], it does not solve the problem of finding the correct source of the communication behind the NAT. The correct approach would be to either place a second probe to a location where the internal addresses are still visible or to export the information about translation to the collector and pair it with existing flow records.

2.4 Flow Monitoring Process

This section aims to describe the process of converting raw packets and corresponding packet treatment information to flow records. We show how the flow selection function from Definition 2.5 relates to this process. For the purposes of this thesis, we will consider dedicated probes equipped with a flow exporter software from now on. Although the flow monitoring process in networking devices is similar, it has differences and specifics that are outside the scope of this work. This section provides a generic overview of the flow monitoring process, and performance related details are left for Chapter 5.

The Figure 2.6 shows a common flow monitoring process and its individual parts, which are discussed in the rest of this section. An alternative description of the flow monitoring process can be found in the IPFIX Reference Model [28]. We will refer to this description where appropriate.

2.4.1 Packet Capture

The packet capture ensures that data from the network are made available for processing in the software. Standard Network Interface Cards (NICs) can be used, as well as specialised hardware-accelerated cards. The schema shows a standard NIC performing a CRC check on received packets. The NIC has to be put in a special monitoring mode in order to receive packets with different destination MAC addresses. These packets are passed to the NIC driver, usually through Direct Memory Access (DMA). The driver either passes the received packets to the operating system or provides its own application interface for accessing the packets from user space. The performance of the different approaches differ significantly, and we will discuss it in Chapter 5 in detail.

Each packet needs to be assigned a (preferably unique) timestamp that marks the point in time at which the packet was received. This process is called packet timestamping. The timestamp can be assigned by the NIC, NIC's driver (or OS), or the software application processing the packet. The support for NIC timestamping is provided by some of the hardware-accelerated NICs. As the timestamping in software can be potentially computationally intensive, we will also discuss it in Chapter 5.

The packet treatment information is provided together with the packets by the NIC's driver. Usually, the interface of the NIC on which the packet was captured and the timestamp of the capture is available at least. Networking devices can provide more information such as egress interface, next hop, or autonomous system number of the destination. This information is not directly available when dedicated probes are used, but can be added externally if necessary.

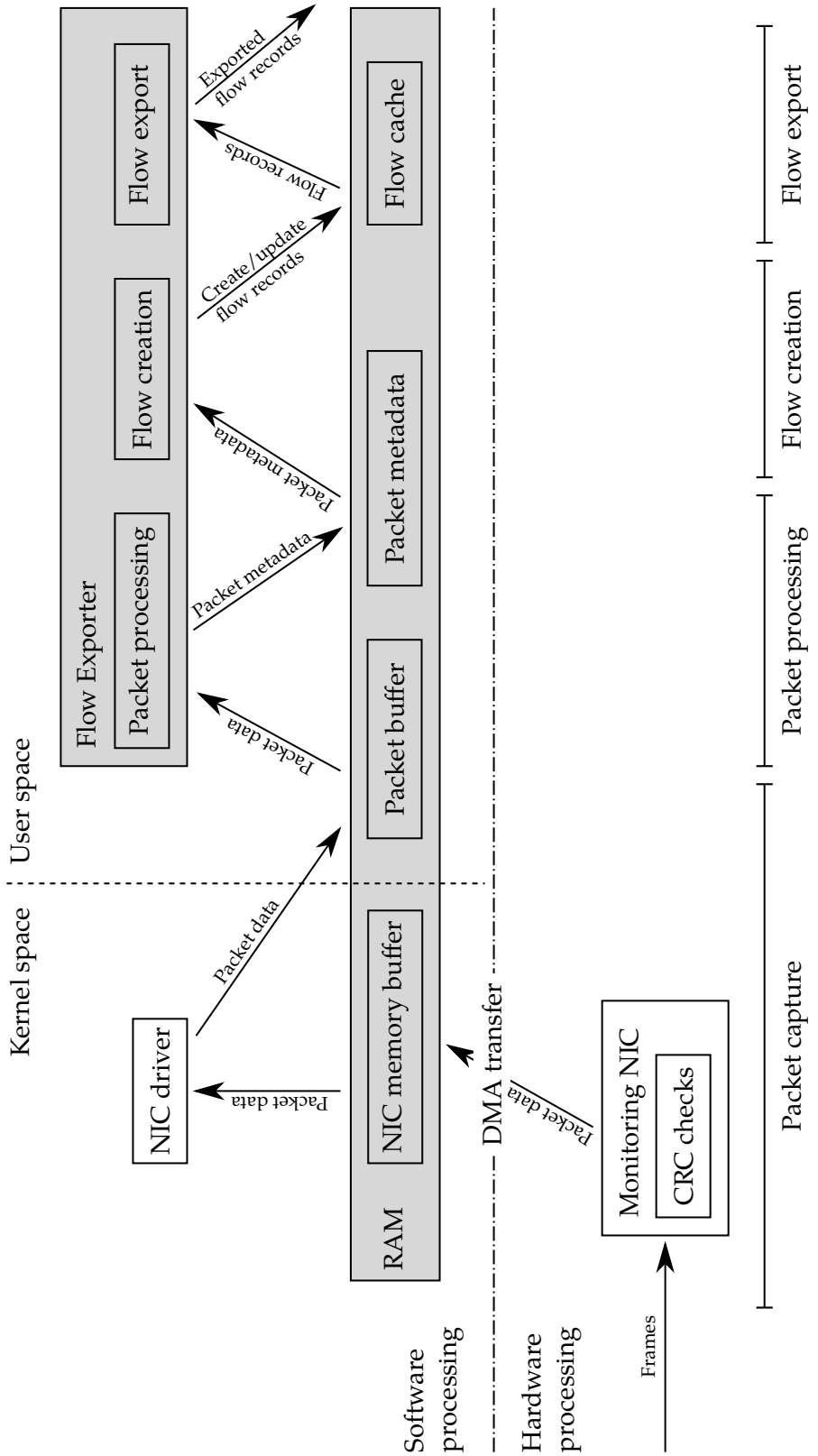


Figure 2.6: Detailed Overview of the Flow Monitoring Process in a Dedicated Probe.

2.4.2 Packet Processing

The task of the packet processing is to extract values of chosen properties of individual packets and corresponding packet treatment information. Attributes such as IP addresses, transport protocol, and ports are used as *flow keys*. The set of used flow keys depend on the applied flow selection function which is used to decide to which flow the packet belongs to. Other attributes of the packets such as TCP flags or the number of bytes are extracted as well for further analysis. We call the extracted properties *packet metadata* or *partial flow records*. The packet metadata are passed to flow creation part of the flow processing where it is used to create new or update existing flow records.

Secondary task of packet processing is packet sampling and packet filtering, as mentioned in [28]. Packet sampling is usually done to reduce the amount of processed data in order to compensate for lack of performance. There are several sampling techniques that can be used as described in [45]. Packet sampling is performed before extracting the metadata. Packet filtering can be performed after the metadata are extracted and is based on the extracted values. It is common to monitor only specific network or a section of the network; therefore the filtering rules are usually based on extracted IP addresses. Both packet sampling and packet filtering can be implemented in hardware-accelerated NICs as well.

2.4.3 Flow Creation

The extracted packet metadata are aggregated to create flow records. The definition of the flow selection function requires all preceding extended packets of the same flow to determine whether a new packet belongs to the particular flow. Since it is not viable to keep all packets of a flow in memory, only selected information is stored in real-world implementations. All active flows have a flow record with all the necessary information stored in a *flow cache*. When a new packet arrives, the flow selection function is called for each stored flow record and the metadata of the new packet to determine to which flow the new packet belongs. If a matching flow record is found, it is updated using the packet metadata (e.g., packet and byte counters are incremented, and the flow end timestamp is updated). If no such record exists, the packet is considered to be the first packet of a new flow and a corresponding flow record is created in the flow cache. Algorithm 2.3 illustrates the flow creation process. It is worth noting that the flow selection function is denoted ϕ as we refer to a concrete implementation here instead of the formal definition.

Testing the extracted packet metadata against each flow record has a linear time complexity with respect to the size of the flow cache. A common optimisation is to compute a hash for each flow record such that it can also be computed for the extracted metadata. The flow cache is then either indexed by using the computed hash [56] or directly implemented as a hash table [59] so that the input and lookup operations have a constant time complexity. Collisions can be solved by a number of common techniques or by prematurely exporting the older colliding flow record. This optimisation is so widely used that its use is often implicit, but it is important to keep in mind that the new packet is still effectively tested against each active flow. Other implementations of flow caches can be found in literature, such as a hypercube flow table by Wang et al. [60].

There are several reasons why a flow record can exit the flow cache:

Inactive timeout occurs when no new packets belonging to the flow arrive for a time interval called *inactive timeout*. This timeout is used to end inactive connections and has a significant influence on flow cache memory requirements. When set too low the inactive timeout can erroneously split idle connections where keep-alive packets are sent in intervals longer than the inactive timeout. The inactive timeout is sometimes called *idle timeout* as well.

```
1: loop
2:   Get new packet P
3:   Extract packet metadata M
4:   Set found = false
5:   for all flow record F in flow cache do
6:     Apply flow selection function φ to F and M
7:     if φ(F, M) = true then
8:       Aggregate M to F
9:       Set found = true;
10:      break
11:    end if
12:  end for
13:  if not found then
14:    Create new flow record F from M
15:    Insert F into flow cache
16:  end if
17: end loop
```

Algorithm 2.3: Construction of Flow Records.

Active timeout occurs when the flow is longer than a time interval called *active timeout*. The reason for active timeout is to keep exported flow information fresh. For example, some SSH connections may be active for months when left open in a UNIX/Linux screen utility, and without the active timeout, the information about the connection would be too late for any practical purpose, such as monitoring the amount of traffic for a given time period. 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 [61].

Natural expiration occurs when the connection is closed normally. This is often implemented for TCP connections where packets with RST or FIN flag indicate the end of the connection.

Resource constraints such as lack of memory can cause flow records to be exported from the flow cache to allow new flows to be inserted. Some implementations of flow caches using hash tables have fixed number of flows that can be stored under single hash. Therefore, when a new flow record needs to be created, one of the existing flow records is usually expired.

Exporter shutdown usually causes all flow records in the flow cache to be exported so that the information about observed packets is not lost. This cause is not often mentioned in the literature; however, we consider it an important one. The exporter must be restarted each time a configuration is changed unless it supports dynamic reconfiguration, which none of the open-source exporters does. Therefore the exported shutdown can happen surprisingly often.

The RFC 5470 [28] considers both timeouts and resource constraints as causes for a flow record to expire but omits natural close of connection and the possibility of exporter shutdown. Flow record expiration setting can significantly influence not only the number of generated flow records, as shown by the authors of [62], but also further flow processing and various flow-based detection methods. Implementation of flow record expiration has an impact on the overall flow monitoring performance. Periodically checking the flow cache for expired flows can create

latency in packet capture that may result in packet loss. Several approaches to this problem are described in the work of Molina et al. [56].

Little attention has been given to the measurement of traffic with fragmented packets. Although the ratio of fragmented packets is decreasing [63], it is still vital to monitor them since it is quite easy for an attacker to hide an attack from intrusion detection systems by fragmenting the attack packets [64]. The RFC 3917 which describes requirements for IP flow information export states the following about packet fragmentation:

> *In case of IP packet fragmentation and depending on the classification scheme, only the zero-offset fragment of a single initial packet might contain sufficient information to classify the packet. Note that this fragment should be the first one generated by the router imposing the fragmentation [RFC791], but might not be the first one observed by the IPFIX device, due to reordering reasons. The metering process may keep state of IP packet fragmentation in order to map fragments that do not contain sufficient header information correctly to flows.*
>
> *Requirements for IP Flow Information Export (IPFIX) [24]*

This means that the flow monitoring system implementing the IPFIX standard is not required to handle fragmented packets. Moreover, neither the Netflow v9 nor the IPFIX flow definitions cover the possibility to assign a packet fragment to the correct flow, because they require the common properties to be derived from the single packet and the corresponding packet treatment. This is the main reason that the Definitions 2.1 and 2.6 allow to derive the common properties from data contained in previous packets of the same flow as well. We assume that the packets arrive in the appropriate order for the purpose of these definitions.

Let us consider what happens when an IPv4 packet is fragmented (the problem is very similar for IPv6). Figure 2.7 illustrates this scenario. When the packet K is fragmented, its payload is distributed between several different packets $(K\#1, K\#2, \ldots, K\#M_K)$. 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. 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 either in the flow cache or during packet capture (e.g., packet reassembly at the OS network stack).

A possible solution that has been successfully deployed in practice is to create temporary flow records for fragmented packets where fragmentation identifiers are part of the flow key instead of transport layer information. Therefore, each fragmented packet has its own temporary flow record. By assigning shorter inactive timeout to these flow records, the temporary flow record can be reinserted into the flow cache when all fragments are received. This allows to efficiently reassemble the original packet with proper transport layer information, which is available since it was present in the first fragment, in the flow cache. The advantage of this solution is that it allows counting the number of reassembled packets as well as the number of packet fragments, which can be used in later analyses.

The flow monitoring implementation based on flow caches of networking devices kept a separate flow for each direction of a network connection. The reason was that each direction usually had different at least ingress and egress interfaces. However, for network monitoring and analysis purposes, it is often useful to be able to access flow records for both communication directions at the same time. For this reason, the bidirectional flow export was proposed in 2005 and resulted in the publication of RFC 5103 [27] in 2008. The document proposes to aggregate

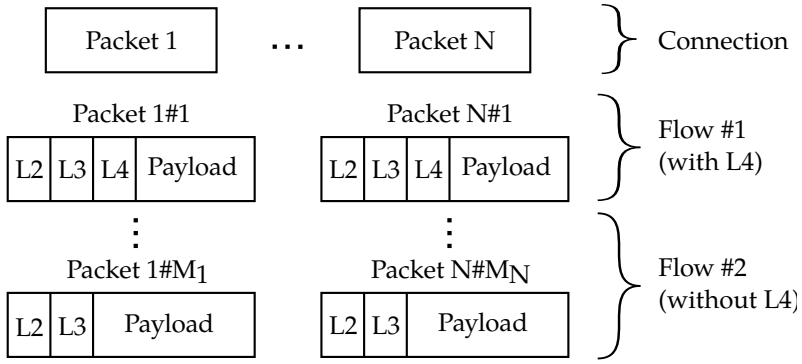


Figure 2.7: Flow Measurement of a Fragmented Connection.

flow records for both forward and reverse directions to a single flow record. Bidirectional flow is called *biflow* and unidirectional flow is called *uniflow*. The flow keys that are associated with a direction (e.g., source IP address) are called *directional flow keys*. When the biflow record is created, some information must be stored for both directions (e.g., number of bytes and packets), and some information is common to both directions (e.g., flow keys). When biflows are used, the corresponding flow records stored in the flow cache become biflow records as well. Note that creation of biflows is permitted by the definitions NetFlow v9 and IPFIX as well as the Definition 2.1. Although we work mostly with the uniflow in the context of this thesis, most of the concepts apply to the biflow as well. We will explicitly mention when there are important differences between uniflow and biflow.

2.4.4 Flow Export

The flow export manages the process of delivering flow records to flow collectors. The process consists of several parts as shown in the Figure 2.8. The flow sampling and filtering process is very similar to packet sampling and filtering described in the Subsection 2.4.2. The primary motivation for flow filtering is that each collector might want to process only a subset of data, e.g., from a particular subnet. The flow sampling is performed when the performance of the collector is not sufficient to process all flow records. The main difference between packet sampling and flow sampling is that the flow sampling always discards whole flow record while the packet sampling can discard some packets of the flow, therefore affecting the created flow record.

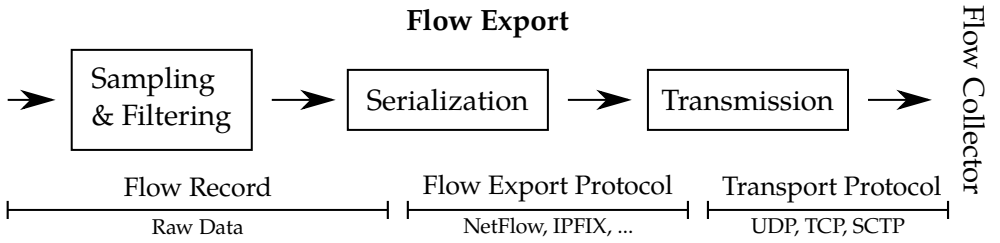


Figure 2.8: Flow Export Schema.

The flow export protocols such as NetFlow or IPFIX describe how to serialise a flow record and how to use different transport protocols such as UDP, TCP, and SCTP to transfer the encoded data to the collector. The NetFlow v5 and v9 protocols are still widely used although the IPFIX protocol is achieving wide deployment since it offers additional features and supports proprietary information elements as well as variable length information elements. Different flow export protocols developed by other vendors can be used as well, as described in Subsection 2.1.1.

The UDP transport protocol is used very often to carry flow records since it was the only one supported by NetFlow. IPFIX specifies the use of TCP and SCTP protocols for flow export as well as the use of the UDP protocol. Although SCTP is mandatory for IPFIX implementations, its real-world usage is hindered by lack of support in software and hardware. When a reliable transport is necessary, TCP is the most often used protocol. For more information about IPFIX transport protocols see RFC 7011 [37]. A comprehensive comparison of IPFIX transport protocols is also provided in [15].

Flow records can be exported in many formats over different transport protocols. With the increasing popularity of big data tools such as Kafka, Elasticsearch, Apache Spark, and Hadoop, the need for a universal and widely supported format is more crucial than ever before. As most of the big data tools support JSON, flow records are often transported and stored in the JSON format. The main advantage of JSON is that the data are stored in a key-value format without the need to specify templates for the data beforehand. The obvious disadvantage is the increase in the message size due to the need to provide a key for every value in every flow record. It is not unusual for the JSON encoded messages to be more than seven times larger than in the IPFIX format.

An important part of flow export is ensuring the security of the exported flow records. The records must reach only the authorised destination without being observed by any third party. Therefore, authorisation, confidentiality, and, preferably, also integrity should be provided. The same applies to flow collectors during the collecting process. More information about flow transmission security is provided in Subsection 2.5.1.

2.5 Flow Data Processing

The aim of this section is to outline the processing of flow data after they are exported to a flow collector. Firstly, we describe the flow collection process in Subsection 2.5.1. The reception of flow data is not a difficult task itself but requires several important choices to be made, such as what flow export protocols will be supported and how to handle unknown information elements. Secondly, the flow data are often stored for future needs. The choice of a flow data storage format is fundamental since the data are written once, read many times and fast searches are required. This process is described in Subsection 2.5.2. Lastly, the flow records are processed and analysed, either by reading the previously stored data or in real time as they arrive from a flow exporter. The flow record processing and analysis is discussed in Subsection 2.5.3. The whole flow data processing schema is shown in Figure 2.9.

2.5.1 Flow Collection

Flow collection is a process of receiving flow records from an exporter performed by a flow collector. The collector and the exporter must choose an appropriate combination of transport protocol and flow export protocol before the data transmission starts as there is no protocol for flow export parameters negotiation. Although the collector can receive data using multiple combinations of protocols at the same time, such feature is not usually implemented due to added complexity.

After the collector validates that a received message is in the expected format, it attempts to parse individual flow records. This process may vary for each flow export protocol; however, the collector always needs to derive correct data types of information elements in order to be able to work with them. The common practice with NetFlow information elements was to hard-code their definitions and extend the code every time it was necessary to add a new element. The extensibility of the IPFIX protocol information elements through the use of Private Enterprise Numbers advocates more dynamic approach to information element handling, although

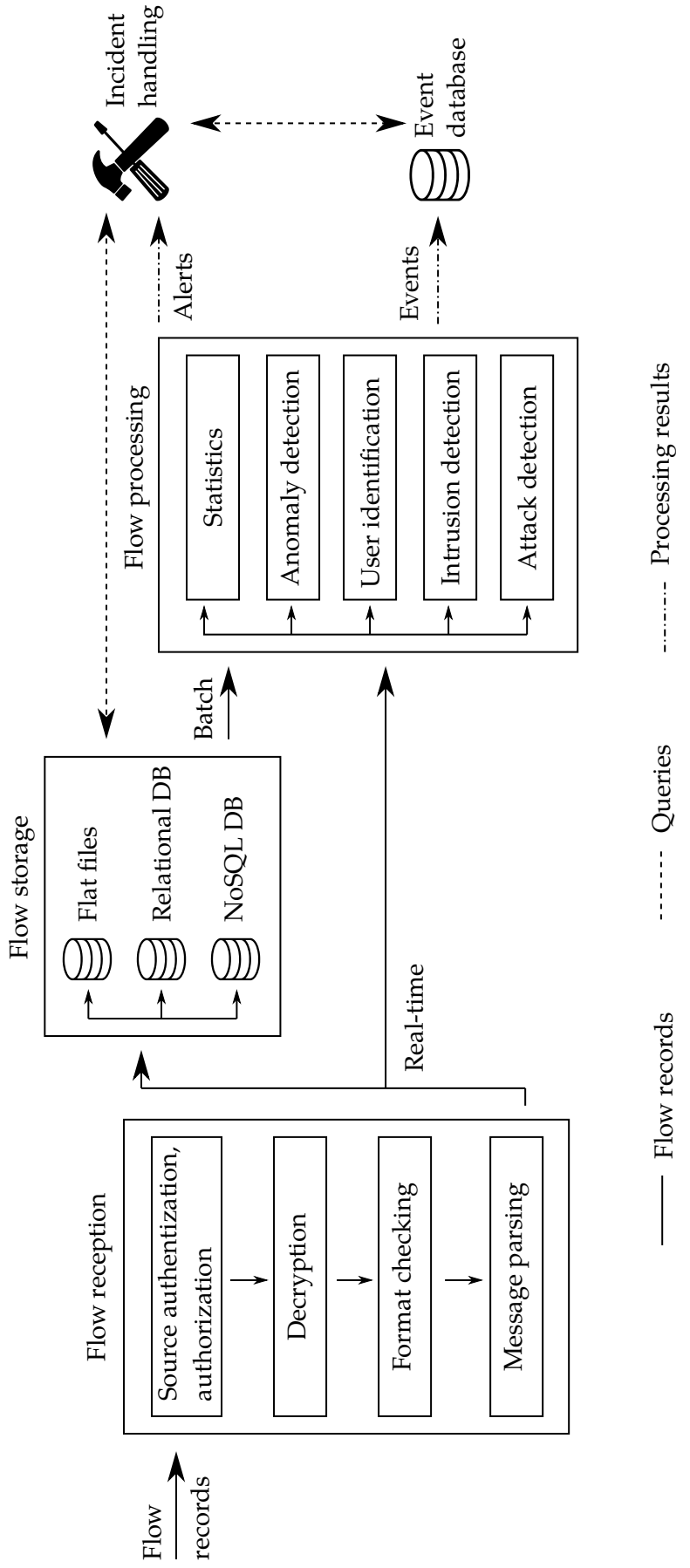


Figure 2.9: Flow Data Processing Schema.

	IP Spoofing	Man-in-the-Middle
UDP	yes	yes
TCP	no	yes
SCTP	no	yes
DTLS/UDP	no	no
TLS/TCP	no	no
DTLS/SCTP	no	no

Table 2.4: Applicability of IP Spoofing and MITM Attack to IPFIX Transport Protocols.

a subset of IPFIX information elements are sometimes hardcoded as well. Since the collectors cannot know all possible information elements that can be sent by exporters, it is necessary to handle new and unknown elements correctly. Although they are usually discarded, it is also possible to process them when their data types can be derived from the information provided by the flow export protocol.

When using NetFlow and IPFIX protocols in combination with the UDP transport protocol, it is necessary to properly manage templates messages containing the definitions of flow record formats. The templates can change during the collection process, e.g., after exporter restart, and it can easily result in incorrectly decoded messages. The IPFIX protocol specification [37] describes how the template messages should be handled and what to do in case of missing templates. Both NetFlow and IPFIX protocols keep track of lost messages by assigning a sequence number to each message. NetFlow v5 increase the sequence number for each transmitted flow record, which allows keeping track of the number of lost flow records. This behaviour was changed in NetFlow v9 so that the sequence number is increased for each transmitted NetFlow v9 packet. It is more difficult to keep track of the number of lost flow records in NetFlow v9, but easy to see how many packets were lost, which can help with determining the cause of the packet loss. The IPFIX protocol increments the sequence number per flow record, just like NetFlow v5. However, when a template is missing, it is not possible to determine the exact number of flow records in a packet. Therefore the collector cannot be sure whether flow records were lost during the transmission or not. This is one of several reasons to use IPFIX over a reliable channel.

An important and often neglected aspect of flow collection is security. The collector needs to authenticate the source of the data to avoid processing of malicious flow records or malformed messages intended to attack the collector itself. There are several ways to authenticate the source of the data. IP addresses can be used to authenticate the flow exporter and the collector itself, or a firewall can be set to allow data from authorised IP addresses. However, this does not prevent attackers with the ability to spoof IP addresses when UDP transport protocol is used. Moreover, the Man-in-the-middle (MITM) attack cannot be prevented without the appropriate use of cryptography. Proper authentication, usually using certificates, is needed to mitigate the MITM attack. The IPFIX protocol requires TLS version 1.1 to be implemented for TCP and DTLS 1.0 to be implemented for STCP and UDP protocols. Moreover, it requires mutual authentication when using TLS or DTLS so that the exporter does not send sensitive data to an unknown target. Table 2.4 shows how IP spoofing and MITM attack are mitigated by use of TLS and DTLS protocols. The advantage of TLS and DTLS protocols is that they provide not only authentication but also integrity and confidentiality. Since an IP address is often considered to be sensitive information, ensuring confidentiality of flow records during transmission should be considered a necessity.

Although the IPFIX protocol requires implementation of TLS and DTLS protocols, not every flow exporter and collector supports it. There are two often used solutions for ensuring the se-

curity of flow record transmission. The first is to build a secure channel between exporters and collectors using external tools, such as virtual private networks or stunnel [65]. The second solution used in networks that are fully under control of the administrator is to place the exporter and collector in a dedicated network (using private IP addresses) so that they can be accessed only from allowed and trusted address range.

2.5.2 Flow Storage

There are many uses for flow data [66, 67]. At first, the data were being used for traffic accounting. Nowadays, flow records are often kept by telecommunications companies to comply with laws requiring them to store communication records. Companies and even individuals also keep flow records of their own traffic so that they can investigate breaches and other security issues ex-post. To store the flow data records for a long period of time, the selected data storage format must be space efficient. Another important aspect of flow data storage is its query performance. The requirements for the query performance differ depending on a particular use case. When overall statistics are required, all flow records need to be accessed. In case of a security breach investigation, filters for particular IP addresses and ports are likely to be used so that it is beneficial for the data storage to have some kind of index built on the data.

There are several types of data storage that are used for flow data. Firstly, the data can be stored in a flat file without indexes. This approach is utilised by popular flow data manipulation frameworks SiLK [68] and NFDUMP [69]. The main advantage is that the format is very simple and the data are efficiently encoded. Moreover, both file formats support compression which allows for more disk space to be saved. Flat files also take advantage of the fact that flow data are never updated so the records can be tightly packed.

A second popular approach is to store flow data in a relational database. In case of NetFlow v5, a single table serves to store all flow records. However, when multiple templates are used for the flow records, such as in NetFlow v9 and IPFIX protocols, the number of tables grows, and the relational model becomes complicated. Although relational databases are designed to support much more features than needed for flow data (e.g., updates or complex joins), they often lack support for network data types such as IP and MAC addresses. Moreover, their performance and disk space requirements are often worse than that of flat files, as shown by Hofstede et al. in [70].

The last approach is to use a NoSQL database to store flow data. Although NoSQL databases do not offer as many features as relational databases, they are meant to be able to handle huge amounts of data. A category of NoSQL databases is columnar databases. Instead of storing each flow record as a single database record, it stores each element of the record in a separate file. The benefit of this approach is that, for example, when a query for a certain IP address is evaluated, only files with IP addresses need to be searched. Other requested elements are read from their own files as necessary after the address is found. This approach reduces the amount of data that needs to be read from the disk; however, it causes more random accesses than when using flat files. We have compared the flat file approach with a NoSQL columnar database in [A2]. Document-oriented databases are also a class of NoSQL databases, and their popularity has been increasing lately. Although the performance of most document-oriented databases is not sufficient to handle flow data from large networks, they are used for their flexibility that allows for building prototype analytics applications very quickly. The advantage of document-oriented databases is that the different structure of flow records is of no concern to the database as it considers each entry to be a generic document. Elasticsearch [71] is an excellent example of a such a database.

The list of used approaches is by no means complete. There are many Big Data analytics frameworks, which are being used for flow data storage and analytics, such as Hadoop [72].

Although it is possible to use almost any data storage for flow data, it is difficult to select the best one, since the criteria differ for each use case. It is for that reason that companies use different flow data storage or develop their own solution in their products.

2.5.3 Flow Processing

A large amount of information about network traffic and communicating parties can be acquired by a flow record analysis. Many intrusion detection systems use flow records as their primary source of data, especially since deep packet inspection is being made difficult due to the increasing ratio of encrypted traffic. The flow data analysis can be performed either using the stored data or in real time.

When the flow record analysis is performed on stored data, the processing is usually done in batches. The flow collector partitions the data according to the time of arrival so that once a partition is finished, it can be processed in a single batch. Five-minute long time windows are often used as this is the default time interval employed by the popular flow processing tool NfSen [73]. The advantage of this approach is that the processing can easily be repeated on the same data, which allows the user to fine-tune the queries and analyse the data on demand. The processing can also be postponed when the system is under heavy load. The disadvantage of batch processing with fixed time intervals is that it introduces delays to data analysis, which can be inconvenient when rapid action is required based on the results.

When the delay induced by the batch processing impedes time-critical applications, stream processing can be used instead. In stream processing, the flow records are passed to the processing immediately after they are received. A good overview of the stream-based flow processing is provided by Jirsik et al. in [74]. There is a number of stream processing systems that are used for big data processing. Čermák et al. performed flow data analysis using three most used distributed stream processing systems in [75], and compared their performance. The results show that the distributed stream processing systems are able to scale to accommodate large workloads such as processing of data from very large networks.

The flow processing can be used to achieve several goals. Firstly, both live and long-term network statistics can be computed. Live statistics are used by administrators to gain an overview of a network situation. This is useful for tracking down problems and optimising network configuration when the changes take effect immediately. The long-term statistics provide useful information for capacity planning. Secondly, flows can be used to gain situational awareness about the network and connected devices [76]. This includes device identification and application classification, which often use machine learning techniques for this purpose. Thirdly, flows are nowadays frequently used in intrusion detection systems to detect attackers on a network, as described by Umer et al. in [67]. Lastly, anomaly detection techniques are being applied to flow data to detect suspicious behaviour which might indicate a problem or attack on the network. There are many anomaly detection techniques as surveyed by Chandola et al. in [77] and significant effort was put to applying these techniques to flow data by various researchers.

Modern machine learning techniques are utilised in flow processing. The most common uses are in traffic classification [A3], user identification [78], and intrusion detection [79]. Although the research in this area is quite extensive, the authors of [80] show that the results are seldom applied in practice and provide several observations explaining the difficulties faced by the implementers. A survey of traffic classification techniques for encrypted traffic is provided in Chapter 6.

2.6 Common Issues

Although flow monitoring is widely used, there are many not widely known aspects that need to be taken into consideration during its deployment. This section describes several issues that can be encountered in any flow monitoring system and that are easily neglected. We draw mainly from our experience of running the flow monitoring infrastructure of CESNET National Research and Education Network in this section. Most issues concern data loss. Either the data are not correctly observed or parsed, are lost during transmission, or are not interpreted correctly. Firstly, we look at the more obvious and readily detectable issue where no data arrive at all. Then we describe the issues that cause data loss which is not easily detected. Lastly, we discuss issues and misunderstandings that cause the data to be incorrectly measured and interpreted. Let us note here, that suitable logging facilities of flow exporter and collector together with a reliable log processing mechanism can help to discover many of the described issues.

2.6.1 Visible Data Loss

The usual cause of data loss is that the monitoring infrastructure is misconfigured. Whether it is due to a misconfiguration at the flow exporter or the flow collector side, the obvious indication of the problem is that no data are stored and no results of the flow data processing, such as statistics, are generated. As the problem is quite easily observed, its cause is easily detected. The best way is to review the whole flow processing setup from packet observation to flow storage and processing to see at which point the data are lost.

The first component to investigate is the packet observation. It is possible that the data are no longer sent to the observation point, possibly due to a failed TAP device or a router misconfiguration. This can be easily confirmed using interface packet counters. Even if the data arrive at the observation point, the flow exporter might be configured to read data from an incorrect interface. When the flow exporter is receiving and processing the data, it is important to audit the flow export settings. The data might be sent to an incorrect destination or using incompatible transport or flow export protocol. When TLS/DTLS is used, the certificates and keys might not match. It is a good practice to verify that the data actually leave the probe, for example by using a tool such as *tcpdump*.

Since the flow probe is often placed in a dedicated network, it is useful to verify that the network is configured to allow the communication between the probe and the flow collector.

The flow collector must be configured to accept data from the flow exporter, which includes a proper setup of certificates and keys. Even when the flow collector receives the data, it might be running using incorrect configuration, and the data might be saved to an unexpected location or not passed to storage and processing at all. A collector may also decide to discard flow records containing unknown information elements. When the flow exporter includes such an element in every record, all flows are dropped.

2.6.2 Unobserved Data Loss

It is impossible to continuously verify that all observed packets are accounted for in the flow records processed by the flow collector. Since each flow record contains information from packets observed at a different time interval, it is not possible to compute the exact number of packets that passed an observation point from the respective flow data. All statistical information about the number of transferred packets is, to a certain level, biased. The only test that can be continuously applied is that the total number of observed packets at the observation point and the flow collector remain in a similar range. Since it is not possible to verify that every packet is observed, some packet of even flows might be lost in the flow monitoring process without notice.

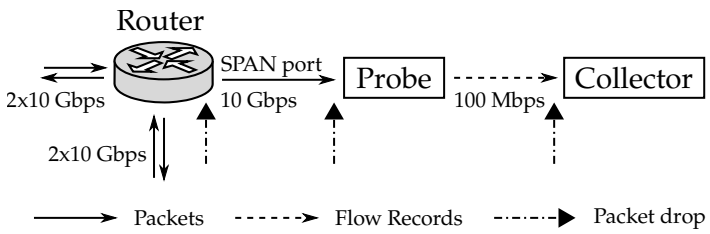


Figure 2.10: Flow Monitoring Packet Drop Schema.

There are many networking protocols in use nowadays. Although all flow exporters support the basic protocols such as IPv4, TCP, UDP, and ICMP, they also need to be able to process many data link layer protocols, such as VLAN, VXLAN, MPLS, TRILL, and others. When a new protocol is used on the network for which the flow exporter is not ready, the flow record cannot be created, and the information is lost. A similar issue involves tunnelling protocols, such as GRE or protocol 41. For example, when the original IP packet is encapsulated in another L3 protocol, it is important to decide whether the flow record should be created from the inner or outer L3 header. Without an additional mechanism to provide more information about the tunnelled traffic, information from one of the headers is lost.

Flow records containing unknown informational elements might be discarded by the flow collector, as described in the previous section. However, when only some of the flow records contain such information elements, the issue is harder to observe. Therefore it is necessary to have a good knowledge of the used flow collector and its behaviour.

Sometimes messages can be longer than the MTU of the link between the flow exporter and flow collector, especially when application layer information is contained in the flow records. In such a case, use of UDP transport protocol might cause data loss as fragmented packets are sometimes blocked by firewalls. Moreover, the packet loss is more probable since by losing a single fragment the entire message is unreconstructable and must be discarded. When such a long messages need to be sent, it is best to use a reliable transport protocol such as TCP or SCTP.

One of the most common reasons for data loss in flow monitoring is that the performance of flow exporter or a flow collector is not high enough to process all data. There are several places where either packets or flow records might be dropped, as shown in Figure 2.10. The first is the source of data for the observation point. When the data for the observation point are provided using router SPAN port, the capacity of the SPAN port must be equal or greater to that of individual links that are measured; otherwise, some packet will be dropped by the router when there is too much traffic. This is not an issue when using a TAP because the TAP always splits a single link or a duplex link to two links of the same type and capacity. The second point where packets are often dropped is the observation point. When the exporter does not process the data before the packet reception buffer is full, the newly arrived or the oldest packets are dropped, based on the implementation of the buffer. The third point where the data might be dropped is on the collector device. When the flow collector does not process data quickly enough, the system packet reception buffers fill up, and new packets are dropped. This is similar to the packet drop on flow probe; however, every packet lost at the collector contains several flow records that were created from many packets. Therefore, loss of packets with complete flow records is much more severe.

The choice of transport protocol has an impact on the packet drop as well. UDP messages are discarded by the collector when it does not process them in a timely manner. However, using a congestion-aware protocol such as TCP and SCTP, the collector might create backpressure on the flow exporter so that it cannot send new messages when it needs to. Therefore, the flow exporter can either drop such messages or wait for the collector to process more data. However,

when the flow exporter does not drop the messages, it creates backpressure on the flow cache, packet processing, packet parsing, and packet reception. This way the slow processing on collector might easily lead to packet drop at the observation point. Another reason for flow exporter not being able to send messages is a slow link between the exporter and collector. When application information is present in the flow records, the required throughput might easily exceed 100 Mbps, especially when the exporter sends data to multiple collectors. Although such slow links are not used very often nowadays, they might still be encountered in practice.

Although packet drop should be avoided at both flow exporter and flow collector sides, sometimes it is necessary to drop packets in a controlled manner, e.g., under a DDoS attack. We recommend creating a packet buffer in the application to which the data are always stored. Then, when the data processing is slow, and the buffer becomes full, the application can decide which packets to drop. The advantage of this approach is that the application is aware of the congestion and can adequately react to it, e.g., by reporting it or limiting the processing functionality to a bare minimum.

A frequent cause of packet loss at the collector side is the shutdown or restart of flow exporter. The flow exporter usually expires all flow records in the flow cache so that they are not lost. However, as the flow cache may contain hundreds of thousands of flow records, exporting them all at one at the highest possible rate (the flow exporter needs to shut down or restart without an unnecessary delay) can overwhelm the flow collector. Therefore, it is best to implement two types of shutdown, emergency and graceful. In the emergency shutdown, the data in the flow cache are simply discarded as they are not considered important. This allows to quickly shut down or restart the exporter, which is beneficial during testing and for important configuration changes. The graceful shutdown should have a configurable flow record export rate. This allows to adapt the capabilities of the flow collector without overwhelming it and limiting the risk of data loss.

2.6.3 Other Issues

There are other problems that can be encountered during flow monitoring apart from the performance issues and data loss. We have already described the problem with counting lost flow records and elements in the Subsection 2.5.1. This section describes the most common of the issues that cause the data to be incomplete or incorrect.

The first issue is encountered by flow exporters that process biflows. To process biflows correctly, the exporter needs to observe both directions of each flow. There are several reasons why it may fail to do so. Only single direction might be connected to the probe, or the network configuration might route the packets asymmetrically. Even if both directions can be observed at the observation point, packets might be received, parsed, and processed in several different queues to speed up the monitoring process. In such a case, it is essential to use a correct algorithm to select queues for packets; otherwise, the packets from opposite directions may end up in different flow caches, and the created biflows will be incomplete. The algorithm must be able to work with all packet headers that precede the headers from which the flow keys are derived.

Both the flow exporter and flow collector must agree on the semantics of the elements of exported flow records. For example, the packet length is usually exported in IPFIX format as a *octetDeltaCount* element. The specification [81] defines it to be the number of octets since the previous report (if any) in incoming packets for this Flow at the Observation Point. The number of octets includes IP header(s) and IP payload. However, some exporters put the total number of octets including the link layer in this field. Moreover, the IPFIX protocol is sometimes used to export information about non-IP flows as well, so that this element is used differently from the specification. Whether the exporter and collector adhere to the specification or not, it is

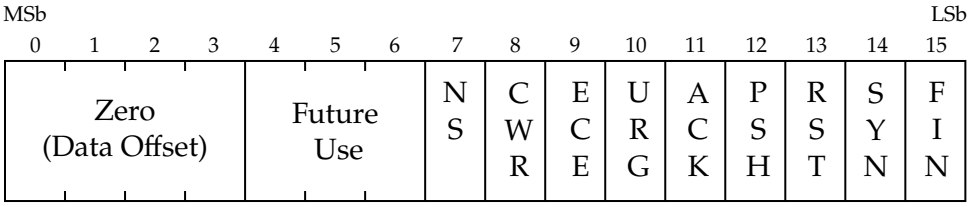


Figure 2.11: TCP control bits export in IPFIX protocol [83].

important that they interpret the semantics equally. In this case, it would be appropriate to set up a different element with different semantics if the total number of octets needs to be reported.

Different exporters can encode flow record elements to integers, strings, and byte arrays of different length. For example, the IPFIX protocol supports *Reduced-Size Encoding* [37] which allows encoding elements using fewer octets than the number specified by [81]. Although this saves resources such as the memory of the flow exporter, network bandwidth, and disk space on the collector, it makes it more difficult to process correctly. When the same element is received by the collector using two different data types, it needs to decide which is going to be used for further processing as maintaining both original data types is cumbersome. The collector usually converts the data to the largest possible data type since it cannot know in advance whether it will receive the element encoded differently in the future and reducing the size of an element may lead to information loss.

The encoding of TCP flags in IPFIX protocol is a good example of issues that can be caused by the *Reduced-size Encoding*. NetFlow v9 defined TCP flags element as one byte, and the IPFIX used single byte as well originally. However, the TCP control bits occupy nine bites in the TCP header [82]; therefore, a change was proposed in 2013 to change the encoding to two bytes. The discussion resulted in RFC 7125 [83] in 2014 with IPFIX supporting the export of TCP flags as a two-byte value, as shown in Figure 2.11. When reduced size encoding is used, only the original byte is exported for backwards compatibility. This presents the authors of flow collectors with a difficult issue. If they support TCP flags as a two-byte value, they must indicate that the received element was only a single byte long. Storing the two-byte value and padding with zeroes is not an option since that would set the ECN nonce sum flag to zero although the collector has no knowledge of its original value.

2.7 Summary

This chapter has explained flow monitoring, its history, common uses, and discussed related technologies that are sometimes confused with flow monitoring. We have shown, that the currently used definitions of a flow do not reflect the reality of real-world flow monitoring deployment. Therefore, we have provided an improved definition that amends the deficiencies. We have also created a formal notation for our definition which allows us to avoid misunderstandings created by the use of informal language. It also allows flow related algorithms to be written more precisely and concisely.

The rest of the chapter has discussed details of the flow monitoring process from the packet capture on a probe to the flow processing on a collector. We have also shown the most common issues that can be encountered while deploying and operating a flow monitoring infrastructure. The most critical issues are caused by incorrect implementation and low performance. Therefore, we investigate the performance of flow monitoring in Chapter 5.

Chapter 3

Application Flow Monitoring

Deep packet inspection (DPI) and basic flow monitoring are frequently used network monitoring approaches nowadays. Although the DPI provides application visibility, detailed examination of every packet is computationally too intensive to be performed on high-speed networks. The basic flow monitoring achieves high performance by processing only packet headers but provides no details about the traffic content. Application flow monitoring is proposed as an attempt to combine DPI accuracy and basic flow monitoring performance. The aim of this chapter is to provide complete information about application flow monitoring.

The contribution of this chapter is threefold. Firstly, it provides an overview of the current state of application flow monitoring. Secondly, it defines consistent terminology and provides a practical definition of application flow and application flow record. The third contribution is an experimental study of an HTTP parser design. Despite extensive work, flow exporters generally fall short of performance goals due to extracting application layer data. Constructing efficient protocol parser for in-depth analysis is a challenging and error-prone affair. We designed and evaluated several HTTP protocol parsers representing current state-of-the-art approaches used in today's flow exporters. We show the packet rates achieved by respective parsers, including the throughput decrease (performance implications of application parser) which is of the utmost importance for high-speed deployments. The presented results provide researchers and network operators with important insight into application flow monitoring performance.

The paper included in this chapter is [A4], another paper related to this chapter is [A5].

The organisation of this chapter is as follows:

- *Section 3.1 provides motivation for processing of the application layer in the flow monitoring.*
- *Section 3.2 outlines the state-of-the-art in the field of application flow monitoring.*
- *Section 3.3 defines necessary terminology and provides definitions of application flow and application flow record.*
- *Section 3.4 describes how the application layer processing affects the flow monitoring.*
- *Section 3.5 is a case study of an HTTP parser design and of the effect of processing the HTTP protocol on the flow monitoring process.*
- *Section 3.6 summarizes the chapter.*

3.1 Motivation

The number of different applications communicating over the Internet is ever increasing, and so is the need for application-aware network monitoring. However, building network monitoring systems is always a compromise between accuracy and performance. The more detailed the information processing, the more accurate the monitoring system is. Unfortunately, a thorough examination of the traffic is computationally expensive [84, 85]. Application flow monitoring is a network monitoring approach created to exploit the benefits of deep packet inspection (DPI). Integration of the DPI into flow monitoring allows for information aggregation, which provides better performance than the DPI alone.

Application flow monitoring is a subset of flow monitoring as described in the Chapter 2 and all provided definitions hold for it as well. The reason to treat application flow monitoring as a special case is that processing application layer introduces specific issues which require particular attention. Therefore, we distinguish basic flow monitoring (flow keys and values of other properties are extracted only from the link, network, and transport layer headers) and application flow monitoring as two distinct parts of flow monitoring.

The application flow monitoring usually utilises two main concepts: application identification (also known as traffic classification) and application visibility. A complete definition of application flow monitoring is provided in Section 3.3. The application identification amounts to recognising the application protocol of a particular flow. The type of application is usually added as a single field to the exported flow record. The application visibility provides more information about the information carried by the application protocol itself. Application identification is a prerequisite for the application visibility. However, application identification can be performed with use of machine learning techniques even without observing packet payloads.

This chapter describes the differences between basic flow monitoring and application flow monitoring that have to be taken into consideration when the application flow monitoring process is designed and deployed. The most important part of application visibility is the design of application parsers. To illustrate the complexity of the application parser design, we propose and discuss several designs of HTTP protocol parsers at the end of this chapter.

The benefits of application flow monitoring are discussed extensively in Chapter 4.

3.2 Related Work

The use of machine learning techniques for traffic classification has attracted many researchers, as demonstrated by [86, 87, 88]. A complete survey of the used techniques and results is out of the scope of this work; however, the methods of application identification in encrypted traffic are surveyed in Chapter 6.

3.2.1 Application Parsers

Although application visibility is provided by a variety of commercial products such as dedicated probes, forwarding devices, and firewalls, it does not seem to be as attractive research topic as application identification where various machine learning and pattern matching algorithms can be applied. However, a significant research effort was invested in automating the creation of application parsers. Pang et al. created a language and accompanying parser called *binpac* [89] in 2006. It allows generating application parsers from their declarative description. A slightly different approach was taken by Caballero et al. in 2007. The authors created a tool called *Polygot* [90] which is used to reverse engineer application protocol headers. Similar work was published at the same time by Cui et al. in [91]. They presented a tool called *Discoverer* that could automatically reverse engineering the protocol message formats of an application from its network trace. Davidson et al. introduce a notion of using a higher order attribute grammar [92],

which allows describing the structure of application protocols for which the use of context-free grammar is impractical or impossible. Another framework, called *Spicy* [93], was introduced by Sommer et al. It consists of a format specification language, compiler toolchain, and an API for DPI applications which allows for easy integration of the generated parsers to existing tools.

3.2.2 Application Flow Exporters

There is a number of open-source tools and commercial products that support export of flows including application level information. Most of them already support the IPFIX protocol for the encoding of application information [15], however, there are still a few that add custom elements to the NetFlow v9 protocol, which can cause element collisions and compatibility problems. Table 3.1 shows an overview of the flow exporters discussed in this section.

Vendor	Product	IPFIX	App. ident.	App. visibility
CERT NetSA	YAF	✓	✓	FTP, HTTP, IMAP, RTSP, SIP, SMTP, SSH, DNS, SSL/TLS, IRC, NNTP, POP3, SLP, TFTP, MySQL, DNP3, Modbus and RTP
ntop	nProbe	✓	✓	
ntop	nProbe Pro	✓	✓	HTTP, DHCP, DNS, MySQL, Oracle DB, BGP, IMAP, POP3, SMTP, Radius, Diameter, GTP, S1AP, SSDP, NetBIOS
FlowMon Networks	Flowmon Probe	✓	✓	HTTP, DNS, DHCP, SMB, E-mail, MSSQL, VoIP SIP and other protocols
Cisco	Application Visibility and Control	✓	✓	
Lancope	Stealthwatch FlowSensor	✓	✓	
Palo Alto Networks	Next-Generation Firewall		✓	

Table 3.1: Flow Exporters Supporting Application Flow Monitoring.

Two open-source flow exporters pioneer the application flow monitoring: YAF and nProbe. YAF (Yet Another Flowmeter) [94] was created as a reference implementation of a flow exporter conforming to the IPFIX standard. YAF supports custom rules for application identification [95]. It can match applications by regular expressions in combination with ports, by signatures or by using dynamically loaded plugins for processing packet payloads. Application visibility [96] is supported for flows where application identification succeeded. YAF allows loading plugins that perform the DPI and export the optional information elements using a subTemplateMultiList feature of the IPFIX protocol. Application protocols supported according to the YAF documentation are listed in the Table 3.1.

The nProbe [97] is a flow probe originally created by Luca Deri and published in 2003. Although the nProbe is both flow exporter and flow collector, we focus only on its features as an exporter. Application identification was added to nProbe in 2011 [10]. It is based on the OpenDPI open-source traffic identification library; however, the authors of nProbe improved the library and showed that their version (nDPI) could be used for high-speed traffic identification [98]. The nProbe supports custom flow export format for NetFlow v9 and IPFIX protocols. The user is allowed to configure her own list of templates that is used to transport data to a flow collector. Support for application visibility is provided by the use of optional plugins. However, these plu-

gins are not available for the open-source version of nProbe and can only be used with nProbe Pro version.

There are multiple commercially available probes, forwarding devices, and firewalls that support at least limited application flow monitoring. It is often difficult or impossible to gain detailed information about the level of application identification and visibility supported by commercial devices. The information provided in the rest of this section is therefore only an overview of some of the available products from publicly obtainable information.

The nProbe Pro version is a commercial version of nProbe supporting plugins that provide application visibility. Application protocols supported according to the nProbe documentation are listed in the Table 3.1.

Another flow monitoring probe is the Flowmon Probe [99] from Flowmon Networks. Support for application identification is provided using the libprotoident [100] library and application visibility is provided by multiple application processing plugins. The application visibility is independent of the application identification; therefore each application processing plugin must have its own application identification algorithm for the supported application protocol. The Flowmon Probe exports flow records using the IPFIX protocol, and the application fields are encoded using the Flowmon Networks Private Enterprise Number (PEN).

Cisco provides support for application identification using the Cisco Application Visibility and Control (AVC) [101] technology both in forwarding devices as well as in dedicated probes called NetFlow Generation Appliances. The AVC uses Network-Based Application Recognition 2 (NBAR2) Protocol Library [102] for application identification and NetFlow and IPFIX protocols for flow export. NBAR2 is a deep packet inspection library which uses signatures to classify traffic into categories and subcategories. Although the technology is called application visibility and control, no application visibility is supported by NBAR2; therefore no details from application protocol headers are exported by the AVC.

The Stealthwatch FlowSensor[103] is a flow monitoring probe by Lancope which supports application identification using DPI and behavioural analysis. The flow export using IPFIX protocol is supported as well as older NetFlow protocols.

An example of a firewall with limited application flow monitoring support is the Next-Generation Firewall from Palo Alto Networks [104]. It supports application identification and can export application labels using NetFlow v9 protocol. Similarly to the Cisco AVC, no application visibility apart from the identification is provided.

All presented flow exporters and flow monitoring devices struggle to achieve high throughput to be able to monitor high-speed networks. However, we avoided direct comparison of declared throughput of the flow probes as it highly depends on the nature of the traffic, supported application protocols, and hardware configuration. More information about flow monitoring in high-speed networks is provided in Chapter 5.

3.3 Application Flow Definition

To describe application flow creation process, an application flow definition should be introduced. Before that, we need to clarify what is meant by different flow monitoring types.

Flow monitoring is defined by Definition 2.1. It is a superset of all following flow monitoring types.

Basic flow monitoring is a flow monitoring that does not utilise application information in any way. It is a complement to the application flow monitoring.

Application flow monitoring is a flow monitoring that utilises application information. It is a complement of the basic flow monitoring.

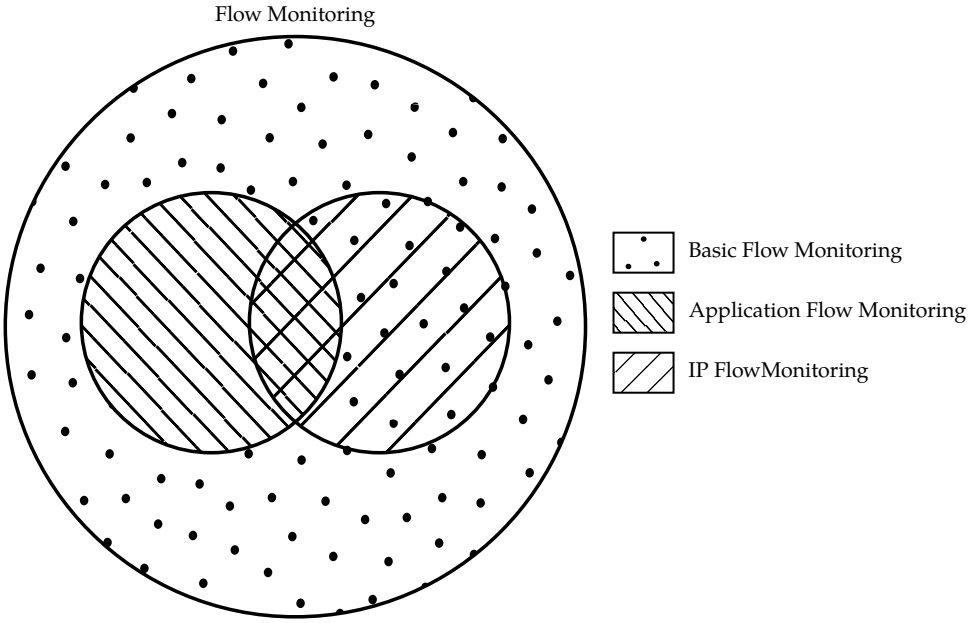


Figure 3.1: Relations between Different Flow Monitoring Types.

IP flow monitoring is often used for flow monitoring that uses information from IP headers in flow keys. It is often used as a synonym for flow monitoring in the literature and is based on NetFlow v9 and IPFIX flow definitions.

Figure 3.1 demonstrates the relations between the described types of flow monitoring. Note that basic flow monitoring and application flow monitoring are complements and IP flow monitoring can be both at the same time.

We have already stated that application flow monitoring is a flow monitoring that uses application information. Let us now consider how that information can be used for creating flows and flow records. Firstly, the goal is to provide application layer information in the flow records. Therefore the flow record needs to be extended with fields containing information extracted from the application layer of the packets of the flow. Secondly, application logic can affect the flow creation process. For example, a flow with continuous HTTP connection can be split into multiple flows based on individual observed HTTP requests. Moreover, the application logic that affects flow creation is not limited to application layer data, although the difference between basic flow monitoring and application flow monitoring is very thin in this case. Consider a scenario where Generic Routing Encapsulation (GRE) is used. As a basic flow, the GRE tunnel would be observed as a single flow. However, the GRE protocol can be considered an application, and by using the semantics of this application, we can split the single flow to many flows based on the traffic that flows through the GRE tunnel. Thirdly, additional information can be inserted from external sources based on the semantics of properties extracted from the packets. For example, geolocation information can be added to the flow records by the probe based on IP addresses of the communicating parties.

Now that the requirements for the application flow monitoring have been specified, we can formulate the application flow definition as follows:

Definition 3.1
An application flow *is a* flow *where either the content of application layer or application logic is used to derive the flow keys.*

The definition specifies what an application flow is. Notice, however, that the definition does not cover all flows that are created by application flow monitoring. The problem is that the definition of application flow cannot specify how the subsequent flow record derived from the application flow will be created. For this reason, we need a definition of application flow record as well:

Definition 3.2
An application flow record *is a* flow record *that contains information derived from either:*

1. *data contained in the application layer of the* flow*, or*
2. *external source of information.*

Application flow monitoring is a subset of flow monitoring where application flows or application flow records are used. Notice that application flow record can be created from standard flow record by extracting information from the application layer. Conversely, standard flow records can be created from application flows. Therefore, either the use of application flows or application flow records are enough to call the related monitoring process *application flow monitoring*.

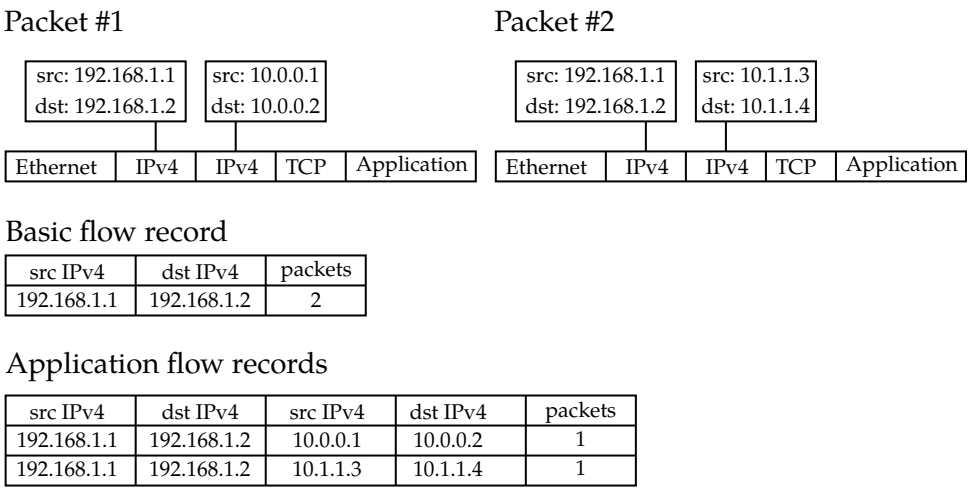
3.4 Creating Application Flow

Application flow monitoring significantly affects the whole flow monitoring process including the data processing on the collector. The most important impact is on the packet processing and flow creation processes of the flow exporter. The packet reception is not affected unless the packets are preprocessed in a hardware-accelerated NIC. This section describes important features of application flow monitoring and their impact on the flow monitoring process.

3.4.1 Packet Processing

The primary goal of packet processing is to extract values of chosen properties of individual packets and packet treatment information. Packet processing in basic flow monitoring can be done using simple packet parsers. These packet parsers only need to be able to process packet headers with simple structures, and the type of the following header is almost always determined by the content of the previous header. However, the type of application in the packet payload cannot usually be recognised from the packet headers. It is not necessary since every client must know where to connect and the server is expected to communicate only with clients using the correct protocol. Therefore, the flow exporters must identify the communicating applications themselves if they are to process the application payloads.

There are three techniques that are used for application identification nowadays. The first and oldest one is based on well-known port numbers assigned and maintained by IANA [105]. Although this method is very simple, fast, and easily implemented, it is not very precise. Since one of the goals of flow monitoring is to provide reliable data for network security, it would be easy for malicious parties to use ephemeral port numbers for their activities to hide their traffic. Moreover, well-known port numbers are often changed legitimately by administrators to improve security. Although this practice is condemned as a case of "security through obscurity", it helps to mitigate a large number of attacks. The second, more reliable, method of application identification is signature based. The application data of each protocol often carry a specific header which can be recognised by pattern matching. Such headers are usually present in the first packets of the connection; therefore the identification happens early enough for the packet processing to be able to process the relevant application data. There are several application identification libraries and related sets of signatures. The paper by Bujlow et al. [106] compares



Basic flow record

src IPv4	dst IPv4	packets
192.168.1.1	192.168.1.2	2

Application flow records

src IPv4	dst IPv4	src IPv4	dst IPv4	packets
192.168.1.1	192.168.1.2	10.0.0.1	10.0.0.2	1
192.168.1.1	192.168.1.2	10.1.1.3	10.1.1.4	1

Figure 3.2: Traffic Encapsulation Example.

the performance of two commercial and four open-source traffic classification tools. Thirdly, machine learning can be used for application identification, as described in Section 3.2. However, these methods often require features that are available only after the flow is exported, such as the number of packets or timing between individual packets. Since machine learning cannot usually recognise the application after first data packet, which often contains the most important information, it is not used for traffic classification by flow exporters.

We have already shown examples of aggregated and non-aggregated basic flow properties in Chapter 2. Properties extracted from application data are mostly aggregated as they are not present in every packet of the flow. Moreover, they are very often present only in a single packet of the flow, e.g., the properties found in HTTP header or parameters of TLS connection during the handshake. The aggregation function applied in such cases is *firstOf*, which means that the first value of the property is used.

Basic flow records are usually created from the first layers of the packets. The most common example is flow monitoring skipping the *link layer* (e.g., Ethernet), extracting information from the *network layer* (e.g., IPv4 or IPv6) and the *transport layer* (e.g., TCP, UDP, ICMP). However, the protocol encapsulation can be much more complicated. For example, protocols such as IP in IP, Generic Routing Encapsulation (GRE), and PPTP allow to create virtual connections and provide a second layer of headers in each packet. Basic flow monitoring usually creates flows from the first layer it encounters. However, when the flow exporter searches for application layer, it should traverse all underlying layers as well. Moreover, the layer from which the flow records are created must be well defined. It is even possible to create flows based on higher layers and therefore split a single basic flow record based on its content. More details on this process are provided in Chapters 4 and 7.

The creation of application layer parsers is a complicated process. Although there are several tools designed for this purpose, as shown in Section 3.2, it is often more efficient to create a particular parser from scratch. We have built several application parsers by hand and using lexical analysis tools for the HTTP protocol and evaluated their performance in Section 3.5.

3.4.2 Flow Creation

The flow creation process is affected in two ways. Firstly, properties derived from application values can be part of a flow key. This is the case of encapsulated protocol where the payload contains more IP headers, as shown in the Figure 3.2. If the addresses of the second IP header are used in flow keys, the basic flow is split into multiple flows, as described in the previous

section. Secondly, the flows can be split when a certain application event occurs. The description of the second case follows.

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. This is an example of the situation in which application logic affects the creation of flows, for which reason they become application flows. Note that the application flow monitoring may affect the number of generated flow records. This needs to be taken into consideration during further processing of the flow records.

Based on the previous example, we can see that the application flow monitoring affects not only flow keys, but the flow expiration as well. Therefore, we add *application event* to the existing flow expiration reasons (inactive timeout, active timeout, natural expiration, resource constraints, exporter termination). The application event expiration is usually implemented when the single connection is reused for multiple events of the same kind. Another example apart from the HTTP pipelining is FTP file transfer, where multiple files are transferred using a single connection, or SMTP connection where multiple messages are delivered at once.

The flow record values extracted from application layer are often present only in a single packet (e.g., HTTP header properties). This causes the information to be lost when the flow is long, and an active timeout is triggered. The new flow record cannot contain the application information since it is not present anymore during its creation. There are two ways to solve this issue. The first is to do nothing during the flow creation and handle the problem during flow data processing. The previous flow records can be looked up, and the necessary information can be extracted from them. The advantage of this method is that statistical queries about application layer data are not affected since the application data are always confined to a single record in this case. The second option is to copy the relevant application information to the new flow record upon active timeout. The information is then easily accessible during flow data processing; however, it is duplicated, and special care needs to be taken when performing statistical queries on the flow data. Either of these options can be taken, but the flow data processing system must be aware of it.

A negative impact of the application flow monitoring on the flow creation process 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 primary negative impacts of such large flow records. Firstly, the flow cache might require much more RAM than basic 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.

3.4.3 Flow Export

The flow export is affected by application flow monitoring in two ways. Due to the larger flow records the amount of exported data increases. The second is that when a template based protocol such as IPFIX is used, the number of templates increases fast with each new combination of supported protocols.

The most important thing that changes for flow export when flow monitoring is applied is that the flow records might be much longer. This can cause congestion of the management link.

When only a part of the data are necessary for flow processing on the collector, the exported data can be shortened. For example, only hostname and fixed length beginning of the URL can be exported for the HTTP protocol. However, since this leads to decreased quality of the exported data, information about the original data, such as original URL length, should be exported as well. The additional data allow the collector to correctly estimate the amount of missing information and apply the necessary algorithm to account for it. Long flow records also have a significant impact on flow record fragmentation when UDP protocol is used. Flow records longer than network interface MTU cannot fit into a single packet and must be fragmented, unless the transport protocol, such as TCP, or flow export protocol handles fragmentation instead. The fragmentation puts additional load on flow collectors and causes increased data loss.

Exporters using template-based protocols, such as IPFIX, need to manage a significant number of templates when application flow monitoring is applied. This can cause performance issues since a template lookup needs to be performed for each flow record. Moreover, flow collectors need to process the templates as well. If the data are stored, for example, in a relational database and each template is represented by a table, a large number of tables can cause the queries to be slow since a union over many tables would need to be performed. Similar problems arise for other forms of flow storage and processing as well.

3.5 Design of an HTTP Parser: A Study

It has been observed that the HTTP protocol became a "new Transmission Control Protocol" (TCP). More and more applications rely on HTTP protocol, e.g., Web 2.0 content, audio and video streaming, and instant messaging. HTTP traffic (TCP port 80) can usually pass through most firewalls and therefore presents a standard way of transporting/tunnelling data. The versatility, ubiquity, and amount of HTTP traffic make it easy for an attacker to hide malicious activities. Missing application layer visibility renders basic flows to be ineffective for HTTP monitoring.

The research presented in this section attempts to answer the following question: What are the impacts of application layer analysis of HTTP protocol on flow exporters and flow monitoring process? The contribution of our work in this section is threefold: Firstly, we have designed and evaluated several HTTP protocol parsers representing current state-of-the-art approaches used in today's flow exporters. Secondly, we have introduced a new flex-based HTTP parser. Thirdly, we report on the throughput decrease (performance implications of application parser) which is of the utmost importance for high-speed deployments.

The rest of this section is organized as follows. Subsection 3.5.1 describes related work. Subsection 3.5.2 contains a description of the HTTP inspection algorithms and the framework that was used to test the algorithms. Subsection 3.5.3 describes the methodology used for HTTP parsers performance comparison. Subsection 3.5.4 presents the performance evaluation of the individual algorithms. Finally, Subsection 3.5.5 contains our conclusions.

3.5.1 Related Work

Application layer protocol parsers are an integral part of many network monitoring tools. We explored the source code of the following frameworks to see how the HTTP parsing is implemented. nProbe uses standard glibc [107] functions like *strncmp* (compare two strings) and *strstr* (locate a substring). YAF uses Perl Compatible Regular Expressions (PCRE) [108] to examine HTTP traffic. Suricata [109] and Snort [110] are both written in C. Suricata uses LibHTP [111] library which does HTTP parsing using custom string functions while Snort does its parsing using glibc functions. httpry [112] is another HTTP logging and information retrieval tool which

is also written in C and uses its own built-in string functions. These HTTP parsers are hand-written.

Another approach is taken by the authors of Bro [113]. They use binpac [89], which is a declarative language with a compiler designed to simplify the task of constructing robust and efficient semantic analysers for complex network protocols. They replaced some of Bro existing analysers (handcrafted in C++) and demonstrated that the generated parsers are as efficient as carefully hand-written ones.

In this section, we try to determine whether these approaches to HTTP parsing can handle large traffic volumes. Besides the above-mentioned approaches, we propose to use the Fast Lexical Analyzer (Flex) [114] to design a new HTTP parser. Flex converts expressions into a lexical analyser that is essentially a deterministic finite automaton that recognises any of the patterns. The algorithm that converts a regular expression directly to deterministic finite automaton is described in [115] and [116].

There are other works that inspect the HTTP protocol headers. In [117] the authors use statistical flow analysis to differentiate traditional HTTP traffic and Web 2.0 applications. In [118] the authors identify HTTP sessions based on flow information. In both cases, a ground truth sample is needed, which is a topic addressed by [119]. In [120] and [121] the authors use DPI to obtain information from the HTTP headers. Our approach is orthogonal to these works since we are interested in extending basic flow records with HTTP data.

3.5.2 Parser Design

HTTP protocol [122] has a number of properties that can be monitored and exported together with basic flow data. The most commonly monitored ones are present in almost every HTTP request or response header. Based on the properties monitored by the state-of-the-art DPI tools we selected the following ones for our parsers: *HTTP method, status code, host, request URI, content type, user agent and referer*. Keeping track of every bidirectional HTTP connection is too resource consuming on high-speed networks. Thus we focus on the evaluation of each individual packet. This approach is common for flow exporters since it is more resistant to resource depletion attacks.

```
1:  if first line contains "HTTP" then
2:    while not end of HTTP header do
3:      for every parsed HTTP field do
4:        if field matches the line then
5:          store the value of the line
6:        end if
7:      end for
8:      move to the next line
9:    end while
10:   return HTTP packet
11: else
12:   return not HTTP packet
13: end if
```

Algorithm 3.1: *strcmp*.

```
1:  if first line contains "HTTP/x.y" then
2:    for all PCRE rules do
3:      if rule matches then
4:        store the matched value
5:      end if
6:    end for
7:    return HTTP packet
8:  else
9:    return not HTTP packet
10: end if
```

Algorithm 3.2: *pcre*.

We implemented and evaluated three different types of parsing algorithms. The first algorithm (*strcmp* approach) loops the HTTP header line by line and searches each line for given fields. It uses standard glibc string functions like *memchr*, *memmem* and *strncmp*. The simplified pseudocode is shown in Algorithm 3.1. The second algorithm (*pcre* approach) uses several regu-

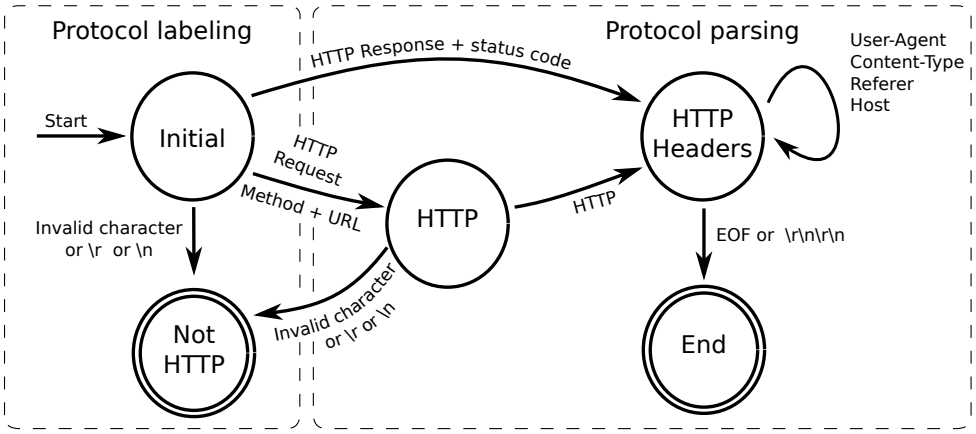


Figure 3.3: *flex* Algorithm Schema.

lar expressions taken from YAF to search the packet for specific patterns indicating HTTP header fields. The pseudocode for the *pcre* algorithm is shown in Algorithm 3.2. We designed the third algorithm (*flex* approach) to handle each packet as a long string. It uses finite automaton to find required HTTP fields, and the Flex lexer is used to process the packets. The automaton design is shown in Figure 3.3.

Since the Flex is a generic tool, its initialisation before scanning each packet is quite an expensive operation. Therefore we decided to remove all unnecessary dynamic memory allocations and costly initialisations to see whether the performance can be increased. We named the new version *optimized flex*. The disadvantage of Flex is that it has to keep the data in its own writeable buffer. Therefore the received data must be copied to such a buffer, which adds to the processing costs significantly. The advantage of the *flex* parser is its simple maintenance and extension possibilities. The framework can be modified to parse any other application layer protocol just by changing the set of regular expression rules. The *strcmp* parser would have to be rewritten from scratch.

The *strcmp* implementation also offers a space for further improvement. Algorithm 3.3 shows an *optimized strcmp* version of the code that features a better processing logic. The optimised version searches for specific strings by comparing several bytes at once, which is done by casting the character pointer to integer pointer. The number that is compared to the string is computed from ASCII codes of the characters and converted to network byte order. The size of the used integer depends on the length of the string; longer integers offer superior performance.

To focus only on the HTTP parsing algorithms, we decided to let the FlowMon exporter [99] handle the packet preprocessing. We used a benchmarking (input) plugin that reads packets from PCAP file to memory at start-up. Then it supplies the same data continuously for further processing. This approach allows us to focus on benchmarking the algorithms without the necessity of considering the disk I/O operations. We provide the source code of implemented algorithms and used packet traces at our homepage [123].

3.5.3 Evaluation Methodology

In this section, we define a methodology of HTTP protocol parsers evaluation. We focus on parsing performance (number of processed packets per second) of the algorithms described in Section 3.5.2 from several different perspectives.

The first perspective focuses on the performance comparison with respect to analysed traffic structure. The second perspective covers the impact of the number of HTTP fields supported by the parser. The third perspective describes the effect of a Carriage Return (CR or `'\r'`) and a Line Feed (LF or `'\n'`) control characters distribution in the packet payload.

```
1: if payload begins with "HTTP" then
2:    store status code
3:    while not end of HTTP header do
4:       for every parsed response HTTP field do
5:          if line starts with field name then
6:             store the value of the line
7:          end if
8:       end for
9:       move to the next line
10:   end while
11:   return HTTP response packet
12: end if
13: if payload begins with one of GET, HEAD, POST, PUT,
       DELETE, TRACE, CONNECT then
14:   store request URI
15:   while not end of HTTP header do
16:      for every parsed request HTTP field do
17:         if line starts with field name then
18:            store the value of the line
19:         end if
20:      end for
21:      move to the next line
22:   end while
23:   return HTTP request packet
24: end if
25: return not HTTP packet
```

Algorithm 3.3: *Optimized strcmp*.

A common technique of increasing network data processing performance is processing only important part of each packet. Therefore, we perform each of the tests in two configurations. In the first configuration, the parsers are given whole packets. This is achieved by setting the limit on packet size to 1500 bytes, which is the most common maximum transmission unit value on most Ethernet networks. In the second configuration, the parsers are provided with truncated packets of length 384 bytes, which is the minimum packet length recommended for DPI by authors of the YAF exporter [94].

To test the performance of the parsers, we created an HTTP traffic trace (testing dataset). Our requirements on the dataset were as follows: preserve the characteristics of HTTP protocol, reflect various HTTP traffic structures and have no side effects on the flow exporter. In order to meet these requirements, we decided to create a synthetic trace.

The HTTP protocol is a request/response protocol. To preserve the characteristics of HTTP protocol during the testing a random request, response and binary payload packet were captured from the network. To omit the undesirable bias of the measurement only these three packets were used to synthesise test trace. The final test trace consists of 200 packets. In order to reflect various traffic structures, we suggested following ratio:

$$r = \frac{\#\text{request packets} + \#\text{response packets}}{\#\text{all packets}} * 100 \tag{3.1}$$

where $r \in [\,0, 100\,]$ and created a test set for each integer ratio from the interval. Further, we created two packets with the modified payload. One packet contained the CR and LF control

characters only at the very *beginning* of the packet payload, the other one only at the *end*. For both of the modified packets and the *unchanged* packet, the test trace for each integer ratio was created.

Having defined the test trace, we propose the following case studies to cover all evaluation perspectives. The case studies are carried out for both full and truncated packets. Moreover, we measure the performance of the flow exporter without an HTTP parser (*no HTTP* parser). This way we can estimate the performance decline caused by increased application layer visibility.

1. *Performance Comparison*: This case study compares the parsing performance of the implemented parsers. Moreover, we report on the flow exporter performance without an HTTP parser (*no HTTP* parser).

2. *Parsed HTTP Fields Impact*: This case study shows a parser performance with respect to the number of supported HTTP fields. We incrementally add support for new HTTP fields and observe the impact on the parser performance.

3. *Packet Content Effect*: The result of this study presents the influence of the CR and LF control characters position in a packet payload on the parser performance. The test traces containing modified payload packets are used to perform the measurement.

The performance evaluation process employs the benchmarking input plugin (see Subsection 3.5.2) to obtain the number of processed packets per second. In order to avoid influencing the results, the plugin uses a separate thread and CPU core for the accounting. The plugin counts the number of the processed packets in a ten-second interval and then computes the packets per second rate. We have operated the benchmark plugin for fifty seconds for each test trace and computed a number of packets processed and a standard error of the measurement. The parsed HTTP header fields impact and packet content effect were assessed in a similar way. All measurements were conducted on a server with the following configuration: Intel Xeon E5410 CPU at 2.33 GHz, 12 GB 667 MHz DDR2 RAM and Linux kernel 2.6.32 (64 bit).

3.5.4 Parser Evaluation

In this section, we present results of HTTP parser evaluation. First, we describe the parser performance comparison; then we investigate the impact of supported HTTP header fields. Finally, the effect of the packet content on HTTP parsing performance is shown.

Performance Comparison.

This case study uses the standard version of each parser that supports seven HTTP fields. The dataset containing the unmodified payload packets is used, and the parsers are tested both on full and truncated packets. Figure 3.4 shows the result for full packets case study and Figure 3.5 shows performance evaluation for truncated packets.

First we discuss the Figure 3.4. The exporter with *no HTTP* parser is capable of processing more than 11 million packets per second. This result is not influenced by the application data carried in the packet since the data are not accessed by the *no HTTP* parser. Employing event the fastest of the HTTP parsing algorithms the performance drops to the nearly one-half of parsed packets per second. All of the HTTP parsers show the decrease in the performance as the ratio r increases. This is caused by growing amount of request and response packets, which are more time demanding to parse.

The best performance is achieved by *optimized strcmp* parser, which uses application protocol and code level optimisations. The parser takes into account the HTTP header structure, the difference between HTTP request and response headers and looks only for header fields that

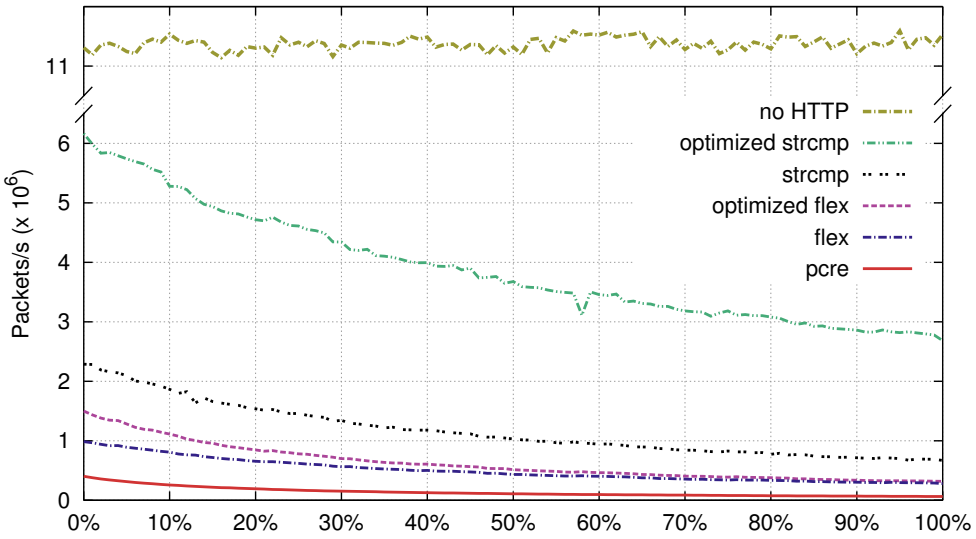


Figure 3.4: Parser Performance Comparison with Respect to HTTP Proportion (0 % - No HTTP, 100 % - Only HTTP Headers) in the Traffic - Full Packets 1500 B.

can be found in the specific header type. The code level optimisations include converting static strings into integers and matching them against several characters at once, which can be done in one processor instruction. The *strcmp* parser performance is the second best, although the throughput is less than half of the *optimized strcmp* parser.

The main difference between *flex* and *optimized flex* parsers is in the automaton initialization process. By rewriting the initialisation process, we achieved slight performance improvement, which is noticeable mainly in the $\langle 0\,\%, 20\,\% \rangle$ interval, where the actual HTTP parsing time is short. There is one other important factor affecting the *flex* parser performance. The flex automaton is designed to work with its own writable buffer since it marks the end of individual parsed tokens directly into the buffer. For this reason, a copy of the packet payload must be created before the actual parsing can start. To measure the impact of the copying, we created another two parser plugins called *empty* and *copy*. First, we measured the flow exporter throughput with *empty* plugin which performs no data parsing, then with *copy* plugin which only copies packet payload to a static buffer. From the results, we estimate the throughput the *optimized flex* parser would have without the memory copying. The performance of the *optimized flex* parser would be about 2.4 million packets per second for 0 % and 0.33 million packets per second for 100 % HTTP packets. This shows that the actual HTTP parsing when compared to *strcmp* parser, is slightly faster for binary payload packets and slower for HTTP header packets.

The performance of the *pcre* parser is the lowest. The PCRE algorithm converts the regular expression to a tree structure and then performs a depth-first search while reading the input string. In case there is no match in the current tree branch, the algorithm backs up and tries another one. Therefore, for a complex regular expression, the pattern matching is not that fast as simple string search using functions like *strcmp*. Another reason why the *pcre* parser is not fast is that it performs all searches on whole packet payload. The other algorithms are processing the data sequentially.

Figure 3.5 shows the results for truncated packets. The *optimized strcmp* and *no HTTP* are only slightly faster since the truncating of the packets has a positive impact on CPU data cache utilisation. The *strcmp* algorithm is flawed since its throughput on HTTP packets deteriorates rapidly. This shows the disadvantage of hand-written parsers, as they are more error-prone than the generated ones. The *pcre* parser performance is almost doubled, as the repeatedly processed data are truncated. Flex based parser also achieve performance increase, since the memory copying costs are reduced for smaller data.

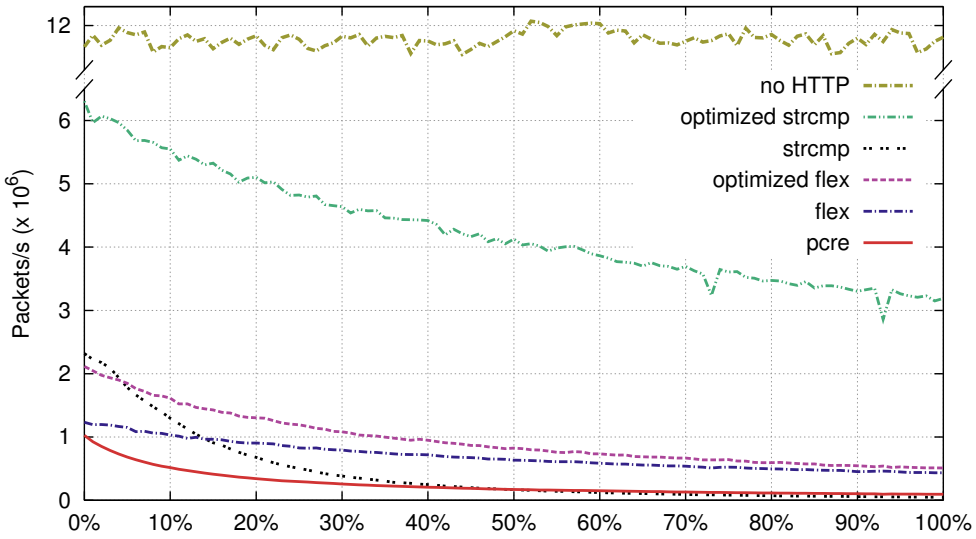


Figure 3.5: Parser Performance Comparison with Respect to HTTP Proportion (0 % - No HTTP, 100 % - Only HTTP Headers) in the Traffic - Truncated Packets 384 B.

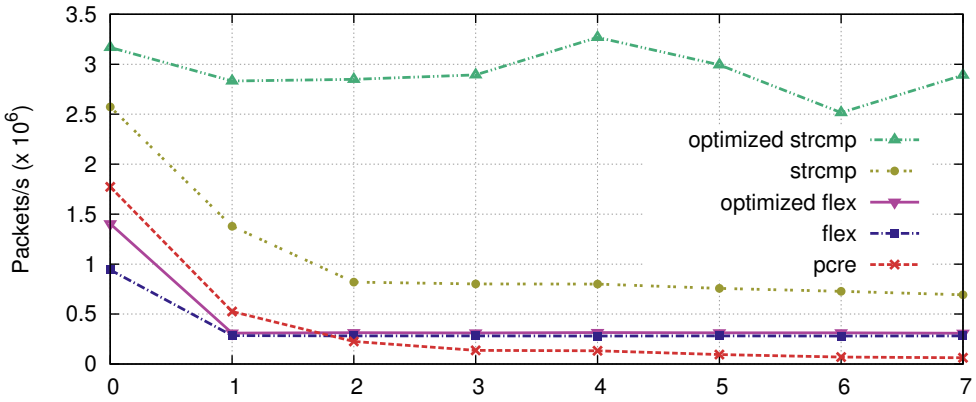


Figure 3.6: An HTTP Parser Throughput for 1500 B Packets; Supported Fields - (0) *none* - HTTP Protocol labeling, (1) *+host*, (2) *+method*, (3) *+status code*, (4) *+request URI*, (5) *+content type*, (6) *+referer*, (7) *+user agent*.

Parsed HTTP Header Fields Impact.

This case study was designed to show the impact of a number of parsed HTTP header fields on the parser performance.

When payload packets are detected, they do not have their content parsed for additional HTTP header fields. Therefore, a test set containing only HTTP request and response packets was used. The case study starts with an empty plugin, that does not parse HTTP header fields and merely labels the HTTP packets. In the next steps, we cumulatively add further header fields to parse until we parse all of the seven supported fields. We run the tests for both full and truncated packets. The average performance of the parsers for each of the added field is shown in Figure 3.6.

Only the request and response packets are parsed, thus the values for the seven fields parsed in the Figure 3.6 correspond to the 100 % packet/s values in the Figure 3.4 and Figure 3.5. For the same reason the parsed packets per second numbers are lower in comparison with the Figure 3.4 and Figure 3.5. The performance of *strcmp* and *pcre* parsers drops with each additional parsed HTTP header field. The performance of the *optimized strcmp* parser slightly fluctuates, as shown in Figure 3.6. An example is the performance increase when adding a *(4) request URI* or a *(3)*

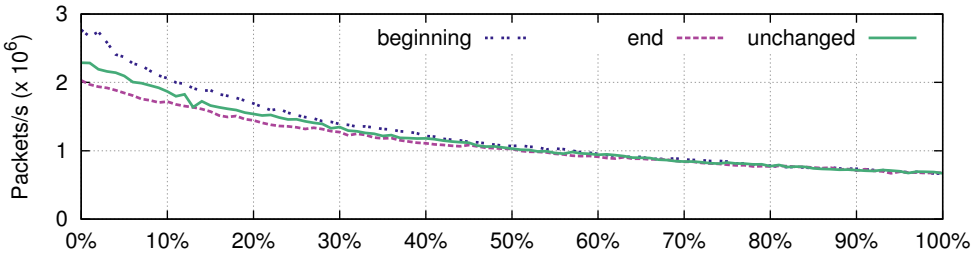


Figure 3.7: Packet Content Effect - Packet Length 1500 B.

status code. It is caused by extra code snippet that extracts the URI so that this line is not processed by the more generic code designed for parsing other header fields. Due to the usage of the finite automaton, the data are always processed in one pass by the flex-based algorithms. Therefore, they retain the same level of performance for all additional fields. This feature could be used to automatically build powerful parsers when a large number of parsed application fields would make it ineffective to create hand-written parsers.

Same as in the previous case study, the parsers processing truncated packets show better performance than the parsers working on full packets.

Packet Content Effect.

This case study investigates the possible effects of the position of the CR and LF control characters in the packet payload on the parser performance. The mentioned ASCII characters represent the end of the line in the HTTP header. Some of the proposed algorithms use these characters as the trigger to stop parsing. Therefore the position of these characters affects the performance of the parsers. The packets with the CRLF characters at the very *beginning* should be parsed faster than the packets having the CRLF at the *end* since the algorithm terminates as soon as it identifies the CRLF characters. The test sets with modified binary payload packets (see Subsection 3.5.3) enables us to compare the algorithms taking into account this perspective.

We used the modified binary payload packets to test the parsers. The parsing algorithms, except the *strcmp* algorithm, show an insignificant difference in their performance for all variants of the modified packets. The *pcre* and *optimized strcmp* parsers do not search for the end of line characters in order to label the packet; therefore this test does not affect them. The flex-based algorithms are not significantly affected since they stop parsing on the first character that is not expected in HTTP header and therefore stop at the first character in any case. The *strcmp* parser depends on the search for end of line characters, which is confirmed by Figure 3.7. The sooner the characters are found, the faster the algorithm terminates. The scenario with truncated packets is different since the performance on *end* dataset is greater than on *unchanged* data. This is caused by removing the end of packet payload together with the end of line character. When the *strcmp* algorithm cannot find the character, it terminates immediately without trying to search the data. Therefore it terminates sooner than on an *unchanged* dataset, where the end of line character is found, and the search continues.

3.5.5 Conclusions

This section has assessed the impacts of HTTP protocol analysis on flow monitoring performance. We implemented the state-of-the-art approaches to HTTP protocol parsing. Moreover, the new flex-based HTTP parser was designed, and its performance was compared to the other approaches.

The evaluation shows that in our case the hand-written and carefully optimised parser performs significantly better than implementations with automated parsing. It also shows that the

new flex-based implementations handle the increasing number of parsed HTTP fields without significant performance loss. Truncating the packets prior to HTTP protocol parsing can increase the parser throughput. The performance comparison of *no HTTP* parser with HTTP parsers shows that providing application visibility is a demanding task. Current approaches to the application protocol parsing may not be effective enough to process high-speed network traffic.

3.6 Summary

This chapter has described the aspects of extending flow monitoring by using information from the application layer. We have argued that application monitoring is necessary to provide information about current network-based cybersecurity threats. Without the application insight, attackers can perform application layer attacks that have no impact visible using the basic flow monitoring. Therefore the application flow monitoring utilises aspects of DPI to provide more fine-grained information about the observed traffic.

We have briefly surveyed the state-of-the-art of the application parsers creation process. There are several approaches that allow creating application parsers from a higher level description, which allows creating more robust and error-prone parsers. However, existing application flow exporters do not often use these approaches and implement their own application parsers. Moreover, although most flow exporters support application identification, only a few of them provide real application visibility.

The existing sources do not use consistent terminology regarding the application flow monitoring. Therefore, we have proposed a workable terminology that captures the current state of the art in the application flow monitoring. We call the flow monitoring without processing the application layer *basic flow monitoring* and differentiate between *IP flow monitoring* which is a term used for any flow monitoring that uses information from the IP layer. We have provided definitions of both *application flow* and *application flow record*, and explained their relation to the flow and flow record definitions provided in Chapter 2.

Once the terminology of application flow has been established, we ventured to describe the application flow monitoring process, particularly the changes that it introduced to the general flow monitoring process described in the previous chapter. It impacts the flow creation on two main levels. Firstly, a new flow expiration reason has been introduced that can be triggered due to an application event. Secondly, the flow records became application flow records as they contain information from the application layer. These changes have an impact on the number of generated flow records as well as on their sizes.

The last contribution of this chapter is a discussion of a design of an HTTP protocol parser. We have shown several approaches to the construction of the parser, implemented them, and evaluated their performance. We have quantified the flow monitoring performance drop caused by each of them and showed that a hand-crafted parser can be highly optimised to provide the best throughput, although it requires an expert programmer to implement it correctly.

Chapter 4

Traffic analysis using Application Flow Monitoring

There are many usages for application traffic monitoring. The common goal is to extend the set of collected information elements and thus provide a deeper understanding of network traffic. This chapter shows several use cases for application flow monitoring, which were published in separate papers. The contribution of this chapter is a summary of the contributions of those papers.

As a first example of traffic analysis with the use of application flow monitoring, we show how information from HTTP headers can be used to detect new classes of attacks on the application layer. This is the most common use case for application flow monitoring.

Basic flow monitoring usually stops after the first header following the IP layer. However, many networks and networking protocols utilise encapsulation to traverse over network segments with unsupported equipment or to allow aggregating of traffic into virtual networks. IPv6 transition mechanisms are a distinctive class of protocols used to provide IPv6 connectivity over IPv4 networks. Most operating systems provide means to deliver IPv6 connectivity even when native IPv6 is not available. Therefore, we extended the basic flow monitoring with information from the IPv6 transition mechanisms and we report on the IPv6 deployment and the use of the transition mechanisms.

Inserting information to the flow records from external sources is also considered to be a variant of application flow monitoring. We show an example of adding geolocation information both in the flow exporter and collector. We show that both implementations scale very well and that the geolocation data can be used for advanced traffic analysis.

Since most flow monitoring experiments are performed on a live network, it is impossible to replicate the results exactly. We show a method of characterising network traffic so that it is possible to compare multiple network traces and search for similarities. This allows us not only to compare different networks but to quantify changes in network profile in time.

The papers included in this chapter are [A6, A7, A8, A9]. Other papers related to this chapter are [A10, A11].

The organisation of this chapter is as follows:

- *Section 4.1 analyses HTTP traffic in a large-scale environment with the use of application flow monitoring. The added value of application flow monitoring is established by detection of several previously undetected attacks.*

- *Section 4.2 reports on IPv6 deployment and the use of transition mechanisms. The basic flow monitoring is extended by the capability of analysing a tunnelled IPv6 traffic.*
- *Section 4.3 shows how flow records can be extended with geolocation information and how this additional information can be employed for network analysis.*
- *Section 4.4 proposes a network traffic characterisation method using flow-based statistics. The characteristics are used to understand and describe the nature of the traffic better and to compare different network traces against each other.*
- *Section 4.5 summarizes the chapter.*

4.1 Security Monitoring of HTTP Traffic Using Extended Flows

HTTP is currently the most widely used protocol which takes up a significant portion of network traffic. Due to its popularity, it is beneficial to have a deeper understanding of HTTP network traffic and its content. From a network security perspective, we would like to know who is accessing our network and what the requested resources are. Patterns in network traffic and outstanding numbers of visited hosts and requested resources would help us to distinguish between legitimate and malicious traffic. The most suitable way of gaining an overview of HTTP traffic in a large-scale network is application flow monitoring, as defined in the previous chapter.

We address two problems in this section. The first one is the lack of an overview of network traffic and insufficient security awareness. This applies especially in heterogeneous networks with distributed administration. Many administrators oversee web servers in their administration and may oversee their neighbourhood, but they are not aware of security threats in the rest of the network. The second problem is to find a suitable set of tools to analyse HTTP traffic and distinguish between legitimate and malicious traffic.

To formalise the scope of our work, we pose two research questions which we shall answer:

(i) What classes of HTTP traffic relevant to security can be observed at the network level and what is their impact on attack detection?

(ii) What is the added value of application flow monitoring compared to basic flow monitoring from a security point of view?

The first question is focused on common types of HTTP traffic that we can observe in the network. We expect to extend the traditional division of network hosts into clients and servers. We focus on the behaviour of hosts regardless of their client/server role: we are particularly interested in detecting security-related patterns produced by attackers, proxies, and crawlers. We search for implications of the presence of particular traffic patterns. We aim to improve malicious traffic detection, filter out false positives, apply better security policy, and set up trustworthy honeypots.

The second question is focused on evaluating the contribution of application flow monitoring with respect to the detection of malicious behaviour in the network. We focus on the detection of patterns in the network traffic that are hard to observe using basic flow. Application flow would make detecting these events straightforward.

Our work contributes to a better understanding of current security threats. For example, we can detect vulnerability scanners and learn about vulnerability itself at the same time.

The answers to both questions are based on an analysis of real traffic in a campus network. We use network application flow monitoring processing the information from HTTP application layer, which provides HTTP elements to application flow records. Although it is possible to extract some information, such as the Server Name Indication, from the HTTPS protocol, this work focuses on the HTTP protocol only.

We also use auxiliary methods to verify and validate the results, i.e., to confirm the membership of detected IP addresses into the proposed behaviour classes. Legitimate sources of traffic should be identifiable, e.g., Googlebot can be identified by a reverse DNS record of the source IP address [124]. We check User-Agents in HTTP traffic to differentiate legitimate and illegitimate traffic, e.g., requests with forged User-Agent. Finally, we check the reachability of traffic destination through search engines to see if it is possible to find the resource without accessing it. We follow the principle of "Google Hacking" [125], a technique used for finding vulnerabilities on web pages indexed by search engines.

This section is divided into six subsections. Subsection 4.1.1 presents previous work in this field. Measurement tools and environment are described in Subsection 4.1.2. The results are pre-

sented in Subsection 4.1.3 and discussed in Subsection 4.1.4. Finally, Subsection 4.1.5 concludes the section.

4.1.1 Related Work

Although research has focused on the analysis of HTTP traffic in recent years, there is only a small number of papers relevant to our work. To the best of our knowledge, we are not aware of any paper dealing with HTTP analysis using a flow-based approach. In addition to this, the motivation of the relevant papers is different, typically focusing on efficient traffic management, with only a minor focus on security, specifically attack detection and prevention. One exception is the work by Perdisci et al. [126]. They employed network-level behavioural clustering of HTTP requests generated by malware. Their motivation was to provide quality input for algorithms that automatically generate network signatures. The second exception is the work by Husák and Čegan [127] in which they use flow-based HTTP analysis to detect users who contacted malicious websites.

Other related papers dealing with HTTP traffic analysis are listed in this paragraph. Augustin and Mellouk [128] published a classification of web applications according to three traffic features: intensity (total volume of exchanged traffic), symmetry (ratio between upstream and downstream traffic) and shape (burstiness over time). Xie et al. [129] addressed reconstructing web surfing behaviour from network packet traces. Xu et al. [130] analysed User-Agents strings captured at the edge of a campus network using an extension of UASparser [131]. They identified operating systems, types of devices (mobile, desktop) and applications (browser, crawler, P2P, automated updates and requests). A similar work by Jin and Choi [132] is motivated by efficient traffic management. Hur and Kim [133] proposed a smartphone traffic classification based on grouping User-Agent fields and extracting common strings.

4.1.2 Measurement Tools and Environment

This section describes the methods and tools used for acquiring the data from our campus network. We shall also provide characteristics of the network> in terms of its size and utilisation.

We performed all measurements on the campus network of Masaryk University. The network has more than 40,000 users, and 15,000 active IP addresses each day in /16 network segment. The network contains both servers and client stations, including proxy servers and a segment of honeypots. Any incoming traffic to the honeypots is by nature suspicious [134], which helps us to recognise malicious network traffic, e.g., HTTP requests on a honeypot web server. Only moderate filtering rules are applied globally to preserve the network neutrality of the academic environment.

Our primary source of data for network traffic analysis is FlowMon [99] probes located throughout the campus network. These probes use NetFlow and IPFIX protocols to export measured traffic as flow records to central collectors [15]. The collected flows are processed by various anomaly and intrusion detection systems, which report incidents to our CSIRT team. This network-centric approach allows us to monitor a complex network with many servers and services managed by different entities as opposed to host-based approaches such as log analysis. It would be challenging to even obtain system logs from every web server in such a varied environment in any case. The monitored links are mostly 1 Gb/s and 10 Gb/s which makes the use of DPI systems impractical, or even impossible in the case of 10 Gb/s links, due to high processing requirements.

To capture data for this section, we used a probe measuring traffic from a 10 Gb/s link which connects the campus to the ISP. We deployed measurement of the HTTP traffic on this probe using an application flow exporter. The exporter software supports the extraction of basic elements from HTTP headers, such as host name, document URL (split into hostname and path),

User-Agent string, and response code. These elements are then exported using the IPFIX protocol as enterprise elements [37]. We used an IPFIXcol [A12] flow collector to receive and store the application flow records.

A basic flow record usually consists of a flow key (4.1) and of other properties providing detailed information about the flow (4.2). The most common flow key contains L3/L4 protocol numbers, IP addresses, and ports. Additional elements usually contain information from packet headers together with statistical counters, e.g., timestamps, number of packets and bytes, and autonomous system numbers.

$$F_{key} = (L3Proto, srcIP, dstIP, \\ L4Proto, srcPort, dstPort) \tag{4.1}$$

$$F_{additional} = (timeStart, timeEnd, packets, octets, \\ TCPflags, ToS, srcAS, dstAS) \tag{4.2}$$

A basic flow record is then a concatenation of F_{key} and $F_{additional}$. The HTTP measurement, described in [A4], adds information from the HTTP application layer (4.3).

$$F_{HTTP} = (hostname, path, userAgent, requestMethod, \\ responseCode, referrer, contentType) \tag{4.3}$$

By concatenating a basic flow record with additional application elements an application flow record is created. In this case, it contains HTTP elements (4.4).

$$F_{app} = F_{key} \cdot F_{additional} \cdot F_{HTTP} \tag{4.4}$$

We measured the F_{app} vector in the environment of our campus network for three weeks over the summer break and three weeks in the semester. Thus, we could compare the amount of overall network traffic and suspicious events over two distinct time periods and traffic loads of the network. The amount of the traffic is lower in the summer break, although still comparable to the rest of the year. However, we assume that the amount of malicious traffic is constant over the year and is more apparent in the summer break.

Table 4.1 shows the size of each dataset in packets, bytes, and flows. Moreover, it contains the number of HTTP requests observed. Although the number of requests is not high compared to the number of flows, there are at least as many responses, which often carry multimedia content. Therefore, the portion of HTTP traffic in bytes is very significant, despite the lower number of requests.

Dataset	Flows	Packets	Bytes	HTTP Req.
Summer 2014	3.733 G	188.953 G	198.362 TB	0.310 G
Spring 2015	6.680 G	552.523 G	604.704 TB	0.720 G

Table 4.1: Datasets.

4.1.3 Results

Four characteristics of network flows were monitored: source IP address, destination IP address, hostname, and HTTP request. We analysed only HTTP traffic incoming to our network. The destination IP addresses were restricted to the /16 IP range of Masaryk University, while the source

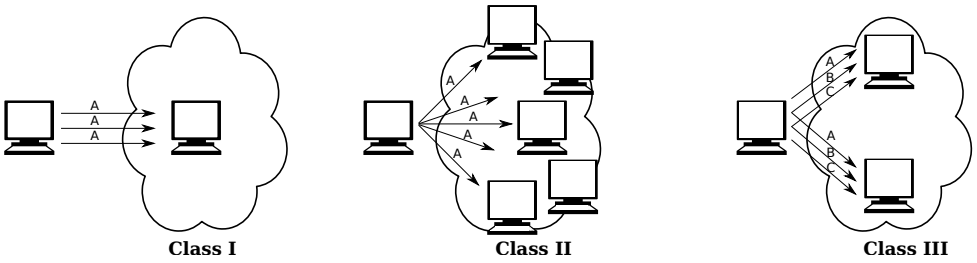


Figure 4.1: Selected Classes of HTTP Traffic.

IP addresses were restricted to all other IP ranges. In some selected cases, we also evaluated the HTTP requests in opposite direction, i.e., source IP was in our network range and the destination IP was not. We used the hostnames and destination IP addresses interchangeably during the analysis due to a negligible amount of multi-hosting in our network. Overall, we observed HTTP traffic sent to almost 500 web servers, i.e., approximately 3 % of all hosts in our network. IPv6 traffic and measurement artefacts were observed in the measurement but were omitted from the analysis due to their negligible impact on results.

The network traffic was analysed over five-minute and one-day intervals. A five-minute interval is the default granularity of data capture in network flow monitoring, i.e., one output file contains records of five minutes of network traffic. It allowed us to observe local anomalies, not only anomalies in the overall sample traffic. The one-day interval allowed us to compare anomalies over different days and discover repeated events and typical patterns. The one-day time window was chosen as it is long enough to contain an anomaly, while it is short enough with respect to the processing time of analysis. The statistics presented in this section are based on an analysis of one-day intervals. However, for practical reasons, the five-minute interval is more convenient for detecting malicious events. We checked that the observed events could be detected even in the shorter time window.

We classified the network traffic according to the three main parameters of an application flow record: guest (source IP address), host (destination IP address or hostname), and HTTP request. We were interested in the cardinality of the relation between the parameters rather than values of the parameters. A high occurrence of flows sharing one or more parameters would suggest unusual activity in the network and is key for the classification. The number of flows which share the three main parameters is, therefore, an additional parameter for classification. The parameters and selected classes are presented in Table 4.2.

	#Guests	#Host	#HTTP Requests	#Flows
Class I	1	1	1	n
Class II	1	n	1	n
Class III	1	>1	n	>n

Table 4.2: Traffic Classes.

We were particularly interested in the results of three queries. First, how many similar requests were demanded by one source IP to one destination IP? Second, how many destination IP addresses were contacted by one source IP address with the same request? Third, how many requests were demanded by one source IP to one destination IP? Each query is represented by a class, as shown in Table 4.2. The classes correspond to patterns in the network traffic as depicted in Figure 4.1. A detailed description of each query and class can be found in the following subsections.

There are many other possible classes, although we found them to not be relevant from a security perspective. For example, many guests accessing a single host with a request suggests

the popularity of a resource at the host. Such classes, however, could be interesting for general traffic classification, not security analysis.

Class I: Repeated Requests

The first query asks for repeated requests between one guest and one host. We measured the number of occurrences of each triple, consisting of source IP, destination IP, and requested URL. Our assumption was that the repetition itself is common, but outstanding numbers may point to undesired behaviour. For example, password-protected web services may be exposed to brute-force attacks, which we can observe as a series of similar requests. The repeated request (A) is defined as follows:

$$\begin{aligned} RepeatedRequest(A) \iff A = \{F \mid \forall F, F' : \\ F(srcip) = F'(srcip) \ \wedge \ F(dstip) = F'(dstip) \\ \wedge F(path) = F'(path)\} \text{ and } |A| > threshold \end{aligned}$$

All flows in the set share the same source IP address, destination IP address, and requested HTTP path. The total number of flows in the set is bigger than the threshold.

Sample results are presented in Table 4.3 from one day in summer 2014. As we can see, the numbers of repeated flows reach tens of thousands, which represents outstanding traffic patterns. There is a disproportion in the numbers of flows with the same source IP, destination IP, and HTTP request. While the majority of the triples were unique or repeated several times, a small number of triples were observed to be repeated several thousand times or more. The nature of these repeated requests was revealed by analysing the request. The majority of repeated requests contained a substring such as *admin* or *login*, which suggests that these traffic patterns were caused by brute-force attacks against password-protected web services. Other interesting repeated requests contained the substring *proxy*, which suggests communication between a client and a web-based proxy server. Furthermore, we found other repeated requests accessed large downloadable files, e.g., ISO images of Linux distributions.

Guest	Host	HTTP Path	#Flows
G1	H1	/wp-**login**.php	46,031
G2	H2	/**admin**istrator/index.php	27,965
G3	H2	/**admin**istrator/index.php	27,798
G4	H3	/wp-**login**.php	25,316
G5	H4	/pub/linux/slax/Slax-7.x/7.0.8/slax-Chinese-Simplified-7.0.8-i486.iso	5,921
G6	H5	/**proxy**/lib**proxy**.pac	5,036
G7	H6	/node/	4,286
G8	H4	/pub/linux/slax/Slax-7.x/7.0.8/slax-English-US-7.0.8-i486.zip	4,170
G9	H7	/wp-**login**.php	3,632
G10	H7	/polit/wp-**login**.php	3,632

Table 4.3: Top Repeated Requests from One Day (Dataset Summer 2014).

The second sample of results presented in Table 4.4 was observed in one day in spring 2015. As we can see, there is a significant difference in the number of flows. However, the distribution of various types of the repeated request is similar. One interesting anomaly is the guest G3,

which performed repeated requests on eight different hosts. This guest used different requests on different hosts; however, the number of flows on each host is similar.

Guest	Host	HTTP Path	#Flows
G1	H1	/**proxy**/lib**proxy**.pac	5,147
G2	H2	/pub/linux/fedora/epel/6/x86_64/repodata/e63d28ff5a765b11a7052496f481f31aebab17c28545908d54e65904a1046ec8-filelists.sqlite.bz2	4,244
G3	H3	/senat/studenti/wp-**login**.php	3,992
G3	H4	/**admin**istrator/index.php	3,945
G3	H5	/slovnik/**admin**istrator/index.php	3,934
G3	H6	/**admin**istrator/index.php	3,926
G3	H7	/capv2011/**admin**istrator/index.php	3,924
G3	H8	/index.php	3,921
G3	H9	/**admin**istrator/index.php	3,794
G3	H10	/wp-**login**.php	3,701

Table 4.4: Top Repeated Requests from One Day (Dataset Spring 2015).

As we can see in Table 4.5, which displays statistics for the whole monitored time window, almost half of the detected events were related to using a proxy. Brute-force password attacks were recognised only in 10.6 % of events. On the other hand, brute-force password attacks may generate more flows than other events in this class. They were the only events which generated more than 10,000 flows with the same source IP, destination IP, and HTTP path. However, we have also observed brute-force attacks of approximately a few thousand flows, but targeting multiple hosts over a short period of time.

Subclass	Path regular expression	Portion [%]
Proxy		**49.4**
	.*libproxy.pac	45.0
	.*sviproxy.pac	4.3
	.*proxy.php	0.1
Brute-force		**10.6**
	.*admin.*	6.7
	.*login.*	3.9
Others		**40.0**

Table 4.5: Distribution of Repeated Request.

Class II: Similar Requests on Many Hosts

The second query addresses identical requests from one source IP address to many destination IP addresses. Our assumption was that a guest accesses only a limited number of hosts or at least requests different paths from different hosts. Therefore, a guest accessing a large number of hosts with the same request (with the exception of "/") is suspicious and worth noting. We would also like to point out the similarity between this traffic pattern and network port scanning, e.g., TCP SYN scan [135]. Therefore, we propose the name HTTP scan for this traffic class. HTTP

scan (B) is defined as follows:

$$HTTPScan(B) \iff B = \{F \mid \forall F, F' : F(srcip) = F'(srcip) \wedge F(dstip) \neq F'(dstip) \wedge F(path) = F'(path)\} \text{ and } |B| > threshold$$

All flows in the set share the same source IP address and requested HTTP path, while there are no flows with the same destination IP address. The total number of flows in the set is bigger than the threshold.

First, we analysed only the scans of our monitored network by external hosts. Sample results from one day in summer 2014 are displayed in Table 4.6. For each scan, we state a guest, i.e., the scanner, the requested HTTP path, the number of visited hosts, and the percentage of visited hosts of all hosts in the monitored network. There were two outstanding source IP addresses identified, one accessed almost all the web servers in our network and requested six paths. The second scanner accessed significantly fewer hosts with only one requested path. The other combinations of Source IP address and path, including the "/" request, were not observed to access more than ten hosts a day. We observed up to nineteen events a day in the measurement, including events where a guest requested more than one path. The same paths were also observed to be requested by different guests.

Guest	HTTP Path	#Hosts	%
G1	/myadmin/scripts/setup.php	497	100
G1	/pma/scripts/setup.php	497	100
G1	/w00tw00t.at.blackhats.romanian.anti-sec:)	497	100
G1	/phpmyadmin/scripts/setup.php	495	99
G1	/phpMyAdmin/scripts/setup.php	494	99
G1	/MyAdmin/scripts/setup.php	491	99
G2	/manager/html	118	24

Table 4.6: Top HTTP Scanners from One Day (Dataset Summer 2014).

The number of detected HTTP scans varied from 3 to 19 per day over the monitored time window. Ten scans per day were detected on average, and scanning for multiple paths was common. The scan for six paths, displayed in Table 4.6, was observed from four different IP addresses in one week. The other multiple-path scans did not use more than three paths in one scan.

The second sample results are from one day in spring 2015. We can see an interesting similarity of requests in the first and second classes. The guest G1 was scanning for HTTP paths which also appeared in the results of the previous class. This suggests that the first class includes brute-force attacks and the second class some sort of network reconnaissance. Therefore, we may have observed the two phases of a complex attack.

Second, we searched for the HTTP scans in the opposite direction, i.e., scanning external networks by a host from our network. The detection in this direction is tricky as there are no clear boundaries of scanned networks and it is hard to set an appropriate threshold. In addition to this, even a common user may generate hundreds of similar requests such as */favicon.ico*. Therefore, we do not present any significant results. However, it would be possible to identify a scan by searching for continuously scanned network segments or the longest common prefix of the scanned IP addresses to estimate a scan of a subnet.

Guest	HTTP Path	#Hosts	%
G1	/wordpress/wp-login.php	337	68
G1	/site/wp-login.php	335	67
G1	/wp-login.php	332	67
G1	/blog/wp-login.php	201	40
G2	/bins.php	183	37
G3	/cgi-bin/test-cgi	102	21

Table 4.7: Top HTTP Scanners from One Day (Dataset Spring 2015).

Class III: Varying Multiple Requests on Multiple Hosts

The third query addressed the guests that requested a large number of unique paths on a single host. In this case, we seek guests that requested the majority of content from our hosts. Our assumption is that a large number of different requests from one guest to one host is a legitimate traffic pattern. On the other hand, outstanding numbers worth noting can also be observed. Then we looked for guests that requested an outstanding number of unique paths on more than one host. We assume that only crawlers behave in this way and it is unusual that any other guest would request such a large number of unique URLs on more hosts. The activity of a crawler (C) is defined as follows:

$$\begin{aligned} Crawler\,Candidate(C') \iff C' = \{F \mid \forall F, F' : \\ F(srcip) = F'(srcip) \ \wedge \ F(dstip) = F'(dstip) \\ \wedge F(path) \neq F'(path)\} \text{ and } |C'| > threshold_1 \end{aligned}$$

$$\begin{aligned} Crawler(C) \iff C = \{F \mid \forall C'_i, C'_j : \\ F_i \in C'_i \wedge F_j \in C'_j \wedge F_i(srcip) = F_j(srcip)\} \\ \text{and } |C| > threshold_2 \end{aligned}$$

We detect crawlers as guests that requested many resources from more hosts. First, all flows which have the same source IP address and destination IP address, but distinct requested HTTP paths, are grouped into sets. The first threshold is a minimal number of distinct requests on a single host. Second, we select source IP addresses which appeared in more sets. The second threshold is a minimal number of crawled hosts. The thresholds were selected to include at most 10 % of sets.

In the first phase, we confirmed that the assumed traffic pattern frequently occurs with a varying number of requested URLs. We were not able to unambiguously mark outstanding numbers. In the second phase, we filtered out the guests that did not access more than one host. The remaining guests were characterised by common signs, e.g., they accessed similar sets of hosts and requested a similar number of URLs. We have marked these guests as crawlers and present sample results from a one-day measurement in Tables 4.8 and 4.9.

Out of eleven crawlers detected in one day in summer 2014, four were identified as a well-known MSN search bot. The others were mostly local search engines, i.e., the Czech search engine Seznam (two guests) and social media monitoring tool Sentione (three guests). The two remaining guests were identified as a part of an IntegromeDB crawler collecting biomedical data.

The second sample results are based on a one-day measurement in spring 2015. In the course of this day, we detected a higher activity of crawlers in our network. The recognised crawlers accessed significantly more hosts and generated more unique HTTP requests. We mostly observed well-known crawlers, such as the MSN search bot and two local search engines. One interesting finding was the detection of an unknown guest behaving like a crawler. The guest's

Guest	Domain Name	#Hosts
207.46.13.62	msnbot-207-46-13-62.search.msn.com	7
157.55.39.107	msnbot-157-55-39-107.search.msn.com	6
137.110.244.137	bnserver2.sdsc.edu	4
157.55.39.156	msnbot-157-55-39-6.search.msn.com	4
157.55.39.6	msnbot-157-55-39-156.search.msn.com	4
37.187.28.19	z3.sentione.com	4
137.110.244.139	integromedb-crawler.integromedb.org	3
5.135.154.106	nks02.sentione.com	3
5.135.154.98	nks03.sentione.com	3
77.75.73.32	fulltextrobot-77-75-73-32.seznam.cz	3
77.75.77.17	fulltextrobot-77-75-77-17.seznam.cz	3

Table 4.8: Top HTTP Crawlers from One Day (Dataset Summer 2014).

IP address had no reverse domain name assigned, and its WHOIS record pointed to a hosting company, which indicates a potentially malicious crawler.

Guest	Domain Name	#Hosts
157.55.39.97	msnbot-157-55-39-97.search.msn.com	24
207.46.13.11	msnbot-207-46-13-11.search.msn.com	23
157.55.39.28	msnbot-157-55-39-28.search.msn.com	18
157.55.39.209	msnbot-157-55-39-209.search.msn.com	17
157.55.39.41	msnbot-157-55-39-41.search.msn.com	15
195.113.155.3	severus.mzk.cz	11
77.75.77.123	screenshotgenerator-77-75-77-123.seznam.cz	11
77.75.77.200	screenshotgenerator-77-75-77-200.seznam.cz	11
46.229.164.99	NOT FOUND	10

Table 4.9: Top HTTP Crawlers from One Day (Dataset Spring 2015).

Similarly to HTTP scans, we were looking for the described activity in the opposite direction, i.e., crawling of external web servers by a crawler from our network. Again, the detection in the opposite direction is harder as there no clear thresholds and many potential false positives. The thresholds for crawler detection had to be set considerably higher to avoid false positives. However, no significant results demonstrating the advantages of this method were found.

4.1.4 Discussion

In the previous section, we outlined three relevant classes of network traffic. The classes cover similar traffic patterns and can be characterised by outstanding numbers of observed network flows. The first class revealed several interesting patterns with ambiguous results as it can be split into several subclasses according to the HTTP request used. The second class is assumed to be solely malicious as it contains scanning for application vulnerabilities. The third class is considered mostly legitimate and is assumed to cover the activity of search bots and crawlers. However, not all the bots and crawlers are welcome in the network. Impacts on network security and the confirmation of the assumptions are discussed in this section for each class. There is also an interesting correlation between the first and second class which suggests that the malicious activity may be observed in both classes.

Class I: Brute-Forcing and Proxy Servers

The first class is characterised by a large number of the same HTTP request from one guest to one host. As we can see in the results, several types of network activity are covered by this class. Generally, we cannot distinguish between them by the number of observed flows. The subclasses are easily distinguishable by analysing the requested path.

The first subclass was observed most often and was identified as communication between client and proxy. This subclass can be easily recognised by the substring *proxy* in a requested path. All the detected hosts were legitimate proxy servers in our network. The clients were located in networks of ISPs in our country, so we suppose the clients were legitimate. The implication for network security monitoring, in this case, is the possibility of proxy detection. We are able to detect active proxy servers in the network and identify their clients. Therefore, we should be able to reveal illegitimate proxy servers in the network or the illegitimate use of a proxy by unauthorised clients, e.g., from networks with a bad reputation [136].

The second subclass was not observed often but generated the record number of flows. Paths containing a substring such as *admin* or *login* suggest a request to a resource protected by authentication. Paths ending in *wp-login* indicate the presence of an administrative interface of a well-known content management system WordPress, which is prone to brute-force password attacks [137]. This path was observed very often and the high number of flows, typically over a short period of time, indicates an attack. We accessed the requested URLs and confirmed that the hosts are running WordPress and provide a login page at the requested path. Another well-known content management system, Joomla!, was also a target of brute-force password attacks. The login page of Joomla! is located under a generic-looking path */administrator/index.php* and we have confirmed the presence of Joomla! at the observed URLs.

An interesting conclusion is that the dictionary attacks are rarely accompanied by any other network traffic from the source IP address. There were neither scans nor any other access to the host preceding the attack as is common among SSH brute-force attacks [138]. We suppose this is due to attackers using a "Google Hacking" technique [125]. In this technique, a search engine (e.g., Google, hence the name) is abused to do the reconnaissance for the attacker. The attacker searches a string distinctive for a WordPress login page, and the search engine returns a list of WordPress instances which can be accessed instantly. Several well-aimed Google searches proved this assumption.

Although we can easily identify the two subclasses by the presence of characteristic substrings, the group of the requested path is uncategorized and creates a third subclass. Paths such as */node/* and */wiki/* do not indicate a vulnerable resource. A dynamically generated content was observed on the requested URLs which suggests that the requests were legitimate. Highly repeated requests for URLs of files for downloading can be explained by partitioning the download, e.g., by download managers, and, thus, is legitimate traffic.

Class II: HTTP Scanners

The second class was marked as HTTP scanning. Requested paths suggest that the scanning guests are searching for specific web applications and their vulnerabilities. It is also common that more paths, versions, or applications are probed during one scan. This type of traffic is easily detectable and highly interesting from a network security perspective. We have not observed any traffic that could be mistaken for a scan or marked as a false positive in this class. Even the low threshold of scanned hosts is sufficient for detection of a scanner as not every scanner accesses every host in the network. Fairly good results can be achieved with a threshold set to one-fifth of the number of web servers in the network.

The HTTP scans were further analysed and correlated to TCP SYN scans on port 80. The results confirm the malicious nature of the scans. 46 % of the HTTP scans were preceded or

accompanied by a TCP SYN scan of the full range of our network. The first option is that the scanner first obtains the list of IP addresses with port 80 opened and then scans them with HTTP requests. The delay between the two phases varies from one hour to several days. The second option of HTTP scanning is to send an HTTP request instantly after receiving a response to the TCP SYN packet.

TCP SYN scanning is easily detectable using network flow monitoring [139], but scanners can avoid detection by scanning "low & slow" [140]. We propose using application flow monitoring to increase the detection rate of network scanning and lower the number of false positives. We observed scanners that scanned no more than 5,000 IP addresses out of a /16 network range with a TCP SYN packet and continued HTTP scanning the web servers they found. Naturally, the scanners did not scan all the web servers; they typically found 100–250 hosts. On the other hand, as we stated earlier, more than 100 scanned hosts is enough to identify a scanner, at least in a network containing around 500 web servers. Therefore, we are able to identify a scanner which would otherwise avoid detection due to the high threshold of the TCP SYN scan detection method.

Another interesting property of the HTTP scans is that some of the requested HTTP paths were observed as a target of brute-force attack as well. For example, the scanners were looking for running instances of WordPress by requesting paths ending in *wp-login*. The same paths were subjects of thousands of repeated request, which we consider as a brute-force attack. This leads us to believe that the attackers search for the victim of a brute-force attack using the HTTP scans, just like it is common for attackers to perform a port scan before a brute-force attack against services such as SSH [138].

Class III: Web Crawlers

The third recognised class of HTTP traffic was marked as crawlers. The query had to be further specified due to the interchangeability of crawlers and common guests when measured on a single host. Covering more hosts in the network provided us with a set of crawlers active in our network. The crawlers were further analysed to filter out false positives. Filtering methods consisted of querying reverse DNS records, querying the WHOIS database, and checking User-Agents used in HTTP requests. We confirmed the identified set was populated by legitimate crawlers. The User-Agents in the HTTP headers pointed to well-known search engines such as Googlebot, Bing, and Baidu. Reverse DNS and WHOIS records further confirmed the presumptions by referring to domains names of the search engine owners.

Crawlers are mostly legitimate and welcome in the network [141]. Almost all the crawlers we discovered were confirmed as legitimate. However, there are two reasons why we included them in a security-related analysis. The first reason is the small number of IP addresses that were identified as crawlers using the flow-based method, but we were not able to tell which any further information about them. Missing reverse DNS records or empty User-Agent fields in HTTP queries lead to the suggestion that these IP addresses performed illegitimate crawling, e.g., e-mail harvesting that aims at discovering spam recipients [142].

The second reason of including crawlers in the analysis is the large number of flows they generate. Any potential detection method based on application flow analysis would have to deal with false positive alerts. Legitimate web crawlers can be easily mistaken for malicious traffic due to their omnipresence and a large number of requests. On the other hand, we have to be careful about false negatives, i.e., malicious hosts disguising as crawlers to avoid attention. In addition to this, operators of web crawlers may change the addresses of their bots over time or not publish them at all, which makes creating a whitelist a difficult task. Our method is based on monitoring the behaviour of the potential crawler, which alleviates the possibility of false

positives or false negative detections. The User-Agent field of HTTP application flow record can also be used for validation of the results.

4.1.5 Conclusion

We presented an analysis of HTTP traffic using application flow monitoring in a large campus network. Contrary to previously published analyses, we were the first to focus on the security aspects. We identified three security-related traffic classes: repeated requests, HTTP scans, and web crawlers. Repeated requests were further split into brute-force password attacks and proxies according to the observed HTTP request.

This classification is based on an analysis of selected elements of application flow: source IP, destination IP, and requested URL split into domain and path. Our analysis supported the hypothesis that flow application monitoring with HTTP headers (Layer 7) enables the straightforward detection of current security issues compared to basic flow monitoring performed at Layers 3 and 4. Particularly, the brute-force password attacks and proxies are hard to differentiate using only basic flow monitoring.

Malicious traffic is contained in the classes of HTTP scans and repeated request, in which we were able to identify brute-force attacks against well-known content management systems. Web crawlers represent a class of legitimate traffic that can be mistaken for malicious activity due to its large number of generated flows. However, suspicious crawlers can also be observed. We have also identified the use of proxies as a traffic class. Using a proxy is not malicious by nature, but is interesting for security enforcement. For example, the presence of an unauthorised proxy may violate security policy in the network.

The classification is based on counting the number of application flows which share one or more parameters. The classification can be thus easily converted into a threshold-based detection method. We were able to detect malicious HTTP traffic in the large-scale network and process it from one point with a low amount of false positives. Ten previously undetectable brute-force password attacks and 10 HTTP scans per day were detected on average in our campus network. In addition, previously unknown proxies and web crawlers were observed. The large-scale network monitoring approach proved beneficial particularly for the detection of HTTP scans and web crawlers, which are hardly detectable using only local system logs or local monitoring.

4.2 An Investigation Into Teredo and 6to4 Transition Mechanisms: Traffic Analysis

Despite IPv6 being the standard for several years, its adoption is still in the process [143]. There are several ways of getting IPv6 connectivity, the dual-stack being the preferred one. Most IPv6 studies deal with native IPv6. However, there are other globally used options known as transition mechanisms. They can provide IPv6 connectivity on networks without native IPv6 connectivity enabled or without an IPv6 ready infrastructure.

The transition mechanisms tunnel IPv6 traffic through an IPv4 network. Despite being supported by major operating systems, there is a lack of studies investigating the characteristics of the tunnelled IPv6 traffic. In this context, this section investigates border traffic of the Czech national research and education network operator (CESNET) and attempts to answer the following question: *What are the characteristics of IPv6 transition mechanisms, in terms of their usage, popularity and impact on native IPv4 and IPv6?*

Our research is motivated mainly by the exhaustion of the IPv4 address space and exerting pressure on network operators and content providers to deploy IPv6. The transition mechanisms are used to facilitate the IPv6 adoption. Unfortunately, they introduce extra elements in

the network which add to the complexity and decrease performance and security. As a result, many existing methods for measuring and monitoring large-scale networks become ineffective.

The contribution of our work is threefold: Firstly, we provide an enhanced version of our flow-based IPv6 measurement system prototype, which enables IPv6 visibility in large-scale networks. Secondly, we analyse and show IPv6 transition mechanisms traffic characteristics including a tunnelled one and thirdly, we show how the traffic of IPv6 transition mechanisms has evolved since 2010.

The section is organized as follows. Subsection 4.2.1 outlines related work. Subsection 4.2.2 provides an overview of Teredo and 6to4 transition mechanisms. Subsection 4.2.3 describes the methodology and measurement setup. Subsection 4.2.4 investigates properties of IPv4 traffic carrying IPv6 payload. Subsection 4.2.6 focuses on the characteristics of encapsulated IPv6 traffic. Subsection 4.2.7 evaluates the use of IPv6 tunnelled traffic. Finally, we draw conclusions in Subsection 4.2.8.

4.2.1 Related Work

The most widely used and discussed tunnelling transition mechanisms are Teredo and 6to4. Although there are several studies focusing on performance evaluation of transition mechanisms listed below, the characteristics of the traffic generated by the tunnelling transition mechanisms are not well known.

A study by Aazam et al. [144] provides a performance evaluation and a comparison of Teredo and Intra-Site Automatic Tunnel Addressing Protocol (ISATAP) mechanisms with a focus on certain parameters like throughput, an end to end delay, round trip time and jitter. A study by Zander et al. [145] compares Teredo tunnelling capability and performance with native IPv6 and 6to4 using measurements related to web services. Teredo increases the time needed to fetch web objects compared to IPv4 or native IPv6. The conclusion is that Teredo seems to be limited by a lack of Teredo infrastructure forcing encapsulated packets to travel long distances. Moreover, the throughput is partially limited by the performance of Teredo relay servers.

A study by Bahaman et al. [146] discusses the performance of 6to4 with focus on communication over TCP. It states that the TCP transmission ability is reduced by the use of 6to4. However, it is still suitable for early stages of the transition period.

Other works discuss the impact of transition tunnels on network security. An RFC by Krishnan et al. [147] presents security concerns with recommendations on how to minimise security exposure due to tunnels. It is pointed out that tunnels can have a negative impact on deep packet inspection and that transition mechanisms such as Teredo allow inbound access from the public Internet to a device through an opening created in a network address translation (NAT) device. This increased exposure can be used by attackers to attack a device hidden behind a NAT device effectively. A frequently proposed security practice is to avoid the usage of tunnels at all and deploy other transition schemes such as dual-stack.

Finally, Sarrar et al. [148] provides a brief insight into tunnelled traffic in a study of the world IPv6 day impact on IPv6 traffic. The Teredo and 6to4 transition mechanisms were monitored, and the Teredo was discovered to carry mainly control traffic. The study also showed that IPv6 fragments were responsible for a significant portion of 6to4 traffic. The authors suspect that these fragments were caused by broken software which most likely forgot to take the IPv6 header size into account.

4.2.2 Investigated IPv6 Transition Mechanisms

The IPv6 traffic is usually divided between *native* traffic and *tunneled* traffic. The tunnelled traffic is considered to be the one encapsulated using other protocols, e.g., UDP or IP protocol 41. This division is not necessarily accurate since the traffic that seems to be native IPv6 can, in fact,

originate from a client using some transition mechanism like Teredo or 6to4. To clarify this point, we will differentiate between native IPv6 traffic, encapsulated tunnel traffic (IPv4 traffic containing IPv6 payload) and decapsulated tunnel traffic. The word tunnel might be omitted for the sake of brevity.

Teredo and 6to4 are the two most frequently used transition mechanisms in the CESNET network. Mechanisms like ISATAP, Anything in Anything (AYIYA), and others based on IP protocol 41 (6in4, 6over4) do not contribute to the tunnelled traffic significantly and do not appear in our analysis. Therefore we will not describe them in detail. We did not analyse NAT64 and DNS64 mechanisms since they should appear as native IPv6 traffic on the outside.

Teredo [149] is designed to provide IPv6 connectivity to an endpoint behind a NAT device. It requires two network components for operation: *relays* and *servers*. Teredo servers are used for initialisation of Teredo (Figure 4.2, communication ①), and after that for opening a port on the user's NAT device in case of a communication which is not initialised by the user. Relays are used for routing and bridging the IPv4 and IPv6 networks. Each Teredo endpoint uses a statically configured server and a relay, which can cause increased latency and low throughput in case of a distant server or relay. Teredo uses UDP for packet encapsulation making the traffic harder to identify.

6to4 [150] is only suitable for hosts with a public IPv4 address. It uses encapsulation in IP protocol 41 packets hence it is relatively easy to detect and monitor. The 6to4 relay servers are acting as a bridge between the IPv4 and IPv6 networks. These relays use any-cast prefix 192.88.99.0/24, therefore, the optimal (nearest) relay server should automatically be used for communication.

Figure 4.2 shows traffic between two endpoints (communication ④), one of which uses the Teredo protocol and the other one 6to4. The IPv6 traffic from Teredo client travels part of its path in Teredo tunnel to be later decapsulated on the edge of IPv6 Internet and shortly after that to be encapsulated again, this time by 6to4 to travel the rest of its path over the IPv4 Internet to the network of its destination. Depending on where the observation point is located, the tunnel (either Teredo or 6to4) or native IPv6 traffic can be observed.

4.2.3 Methodology and Measurement Setup

To perform a thorough inspection of tunnelled traffic, we need to decapsulate packet headers of inner packets. We use the same flow-based framework as in [151] which has been further modified to extract more detailed information from tunnelled data. The main part of the framework is a plug-in which replaces input and processing parts of existing flow generator FlowMon Exporter [99].

Every packet is being processed to extract basic flow statistics, and the processing of inner headers continues to the point when previously extracted fields indicate the absence of observed encapsulation types.

Teredo protocol is detected when IPv6 header is found encapsulated in UDP packet, AYIYA is searched for in packets on TCP or UDP port 5072. Other protocols are recognised by IPv6 address format, which is protocol specific, and the 6to4 protocol can be additionally identified by usage of IPv4 anycast address belonging to 6to4 relay. If encapsulation is present, its type and encapsulated IPv6 header fields are used to extend the set of extracted fields and to identify individual flows taking place inside the tunnel.

Since we need to define new elements for application flow records, Internet Protocol Flow Information Export (IPFIX) protocol is used. It allows using *Enterprise Elements* which can extend the application flow records with additional tunnel information. The framework is able to recognise and extract information from Teredo, AYIYA, and other protocols that are based on IP protocol 41 such as ISATAP, 6to4, and 6over4. We provide the source code of the measuring tool under BSD license at the project web page [152].

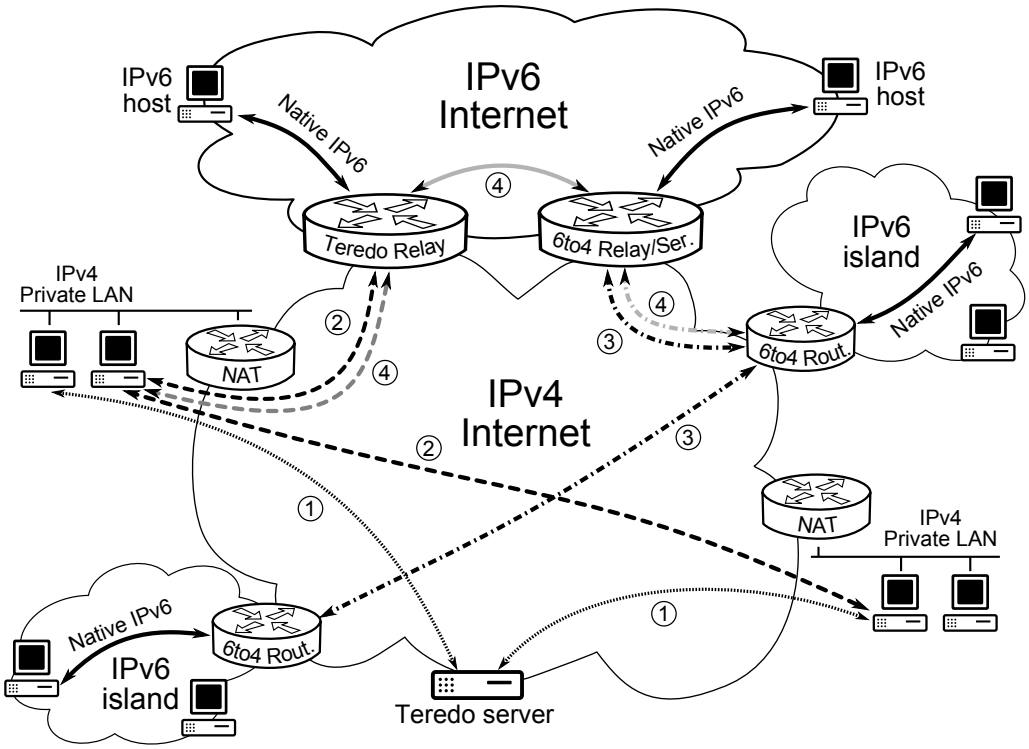


Figure 4.2: Teredo and 6to4 Principles: ① Teredo Start Setup, ② Teredo Traffic Transiting over IPv4 Network, ③ 6to4 Traffic Transiting over IPv4 Network, ④ Communication between Teredo and 6to4 Endpoint.

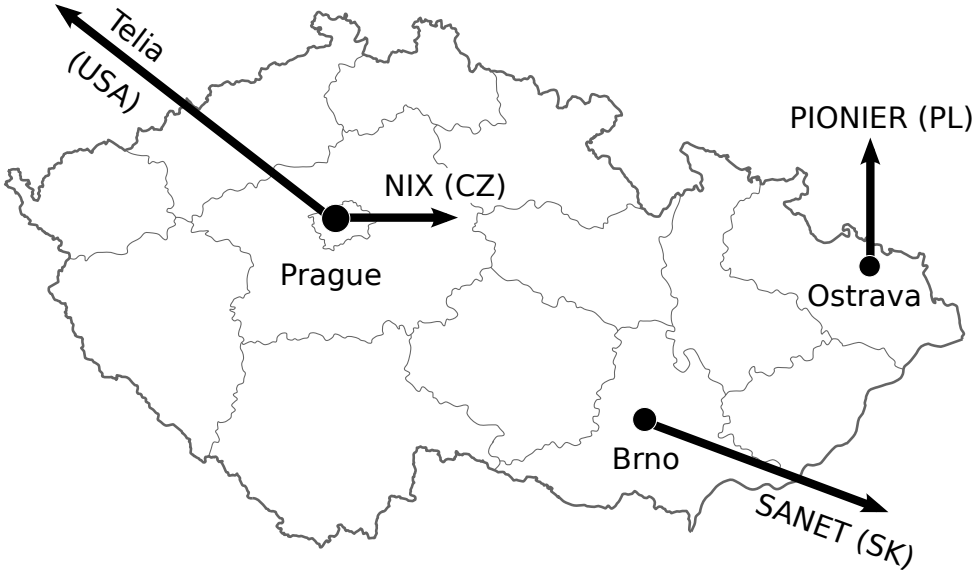


Figure 4.3: CESNET Monitored Links.

Resulting flow data provides us with information about the encapsulated source and destination addresses, ports, and transport protocol, which is a common five-tuple used to distinguish individual flows. We respect this principle and thus have separated the flows encapsulated in the same tunnel based on the value of these elements. Apart from these key elements the framework gives information about *Time to Live* (TTL), *encapsulated HOP limit*, *TCP flags*, and *ICMPv6 type* and *code*, when present. Moreover, additional information about tunnel type is provided,

Probe	Bits/s	Packets/s	Flows/s
Telia	1.65 G	274.9 k	22.1 k
NIX	7.17 G	1072.4 k	26.7 k
PIONIER	0.51 G	75.6 k	2.8 k
SANET	1.87 G	242.3 k	5.3 k

Table 4.10: Observation Points IPFIX Statistics.

including Teredo header and trailer types when present. The framework also newly supports geolocation using MaxMind [153] GeoIP database for both outer and encapsulated addresses.

The data are collected from several observation points located at the borders of CESNET network by passive probes; see Figure 4.3. All measured lines are at least 10 Gbit/s and together transport about 80,000 flows/s during work hours, which results in total traffic of 15.4 Gbit/s. We use IPFIXcol [A12] framework to collect the extended flow data over TCP and to store them. New elements can, therefore, be defined and used without any further difficulties or limitations.

The IPFIX data was collected over one week in January 2013 without the use of any sampling. Table 4.10 shows the average amount of traffic for all observation points. The total amount of stored data took approximately 2,485 GB of disk space. All statistics presented below are based on flow count.

4.2.4 Characteristics of IPv4 Tunnel Traffic

In this section, we describe characteristics of IPv4 traffic containing IPv6 payload. The analysis is based purely on information from IPv4 headers, and extended flow data are only used to identify relevant flows accurately. Three characteristics that can give us insight into tunnelled traffic are addressed. Firstly, we describe TTL values of the various traffic sets, and then we look into a geolocation aspect. Lastly, the basic flow statistics are presented.

Frequency of TTL and HOP Values

We study the distribution of *TTL values* of the observed flows. It is known that some operating systems use specific values, as shown in [154]. Microsoft Windows has the default TTL set to 128. The value of 64 is mostly used by Mac OS X and Linux devices, including devices running Android. We expect that these operating systems form a majority and are therefore the most significant. Figure 4.4 shows the most frequently used TTL values for IPv4 flows carrying IPv6 payload. The TTL values are most frequent near the values set by OS vendors, and the frequency is decreasing rapidly in less than ten hops. Therefore, we assume that most of the packets reach their destination in less than 32 hops. Thus we classify the flows according to their TTL numbers into four significant groups. The Windows traffic seems to be the most frequent one taking 60.3 % of the total, while Linux machines are not present so often with only 23.8 %. Apart from the Windows and Linux ranges, there are devices that set TTL to 255 and 32. Although the 255 are usually Cisco routers, in case of tunnelled traffic we observed that the 6to4 traffic from anycast addresses have TTL set to 255 as well. The portion of the 255 range is 3.8 % and most of it originates from 6to4 relays. TTL numbers 24 and 26 are dominant in the group of values from 1 to 32, which makes 12.2 % of the total number of flows. We discovered that this is caused by a 6to4 tunnel that passes two observation points. The tunnel is heavily used and causes a large portion of tunnelled traffic, which also affects other 6to4 measurements.

Overall, the TTL distribution of IPv4 traffic is different as shown in Figure 4.5. The Linux portion of the traffic is higher than the rest, and the TTL values of 32 and 255 are not as significant. A more detailed examination of the flow records shows that this is caused mainly by a high ratio

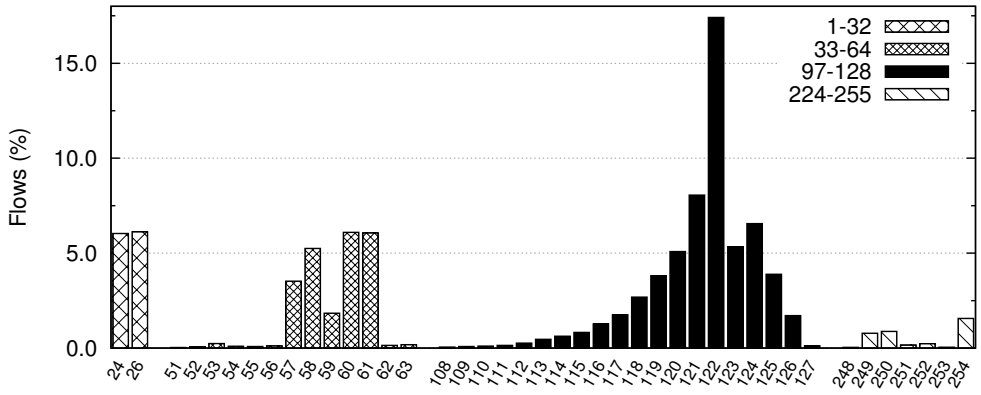


Figure 4.4: TTL Value Distribution of IPv4 Traffic Containing IPv6 Payload.

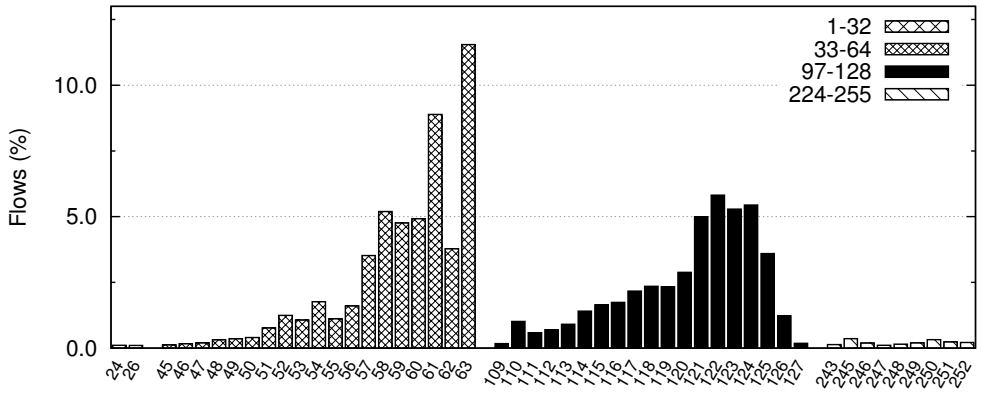


Figure 4.5: TTL Value Distribution of Total Observed IPv4 Traffic.

of HTTP traffic. Even though there are more clients using Windows operating system, most of the web servers are based on Linux, and therefore the responses have TTL less than 64. The DNS protocol shows similar characteristics except that the DNS traffic that we observe at our metering points is mostly generated by recursive domain name servers. Since the Linux DNS servers are the most widely used, they significantly contribute to the traffic generated by Linux machines.

IPv6 uses *HOP limit* instead of TTL. Figure 4.6 shows HOP limit distribution of native and decapsulated IPv6 traffic. Unlike IPv4, the HOP limit of 64 is the most frequent. We assume that Linux based machines use default HOP limit 64 and Windows machines use default HOP limit 128. This setting can be overridden by Stateless Address Autoconfiguration. Therefore, clients in managed networks (e.g., universities) might have the HOP limit set to a different value, regardless of their operating system. We verified this fact on several Linux and Windows-based machines. Due to the significant share of HTTP(S) in IPv6 traffic, a large portion of Windows traffic is expected. Since the share of HOP limit 128 is negligible, we expect that common HOP limit in observed IPv6 networks is set to 64.

Location of IPv4 and IPv6 Endpoints

The second characteristic that we evaluate is *geolocation aspect* of the IPv4 and IPv6 traffic. We focus on data from Telia link only, which connects the CESNET network to the United States. This highlights the differences in geolocation characteristics better. The statistics are computed separately for the incoming and outgoing lines and are shown in Figure 4.7. The IPv4 is more symmetric than IPv6 since the country statistics for both directions are similar. This is normal behaviour since most of the requests initiate a response and the routes are also symmetric.

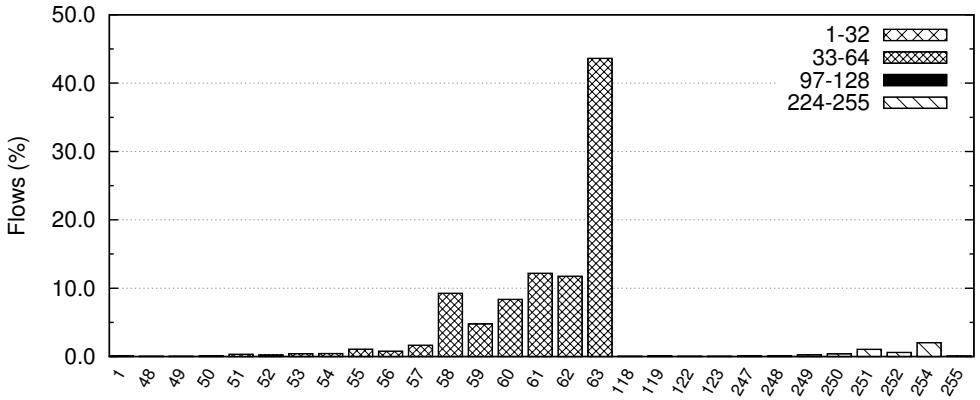
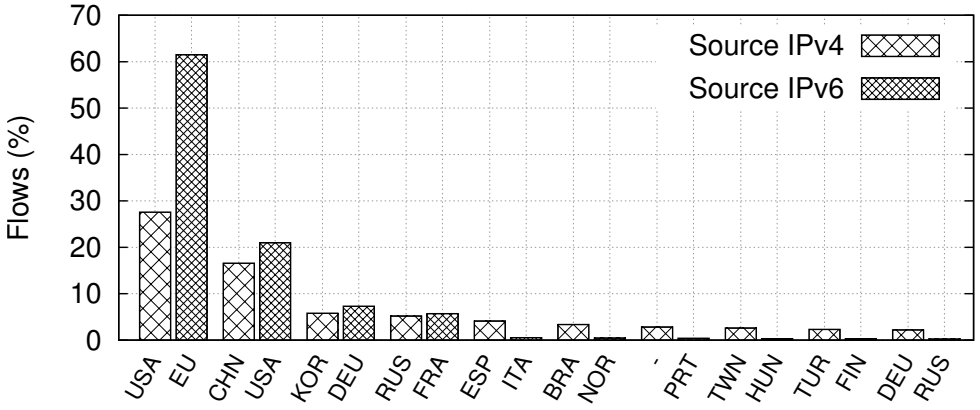
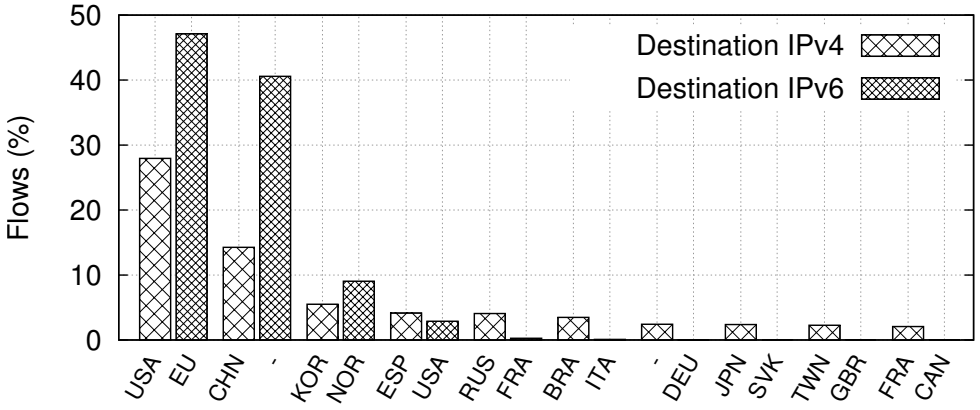


Figure 4.6: HOP Value Distribution of IPv6 Traffic.



(a) Incoming Traffic.



(b) Outgoing Traffic.

Figure 4.7: Top 10 Country Distribution for Native IPv4, IPv6 and Decapsulated IPv6 Addresses.

The IPv6 have different properties. We discovered that the addresses that cannot be geolocated are mostly link-local addresses (fe80::/10) or local-link multicast addresses (ff02::/16). Such addresses should not be routed at all, which indicates that there are routers with erroneous IPv6 configuration. This misconfiguration also causes the asymmetry of the traffic, as such requests cannot be answered.

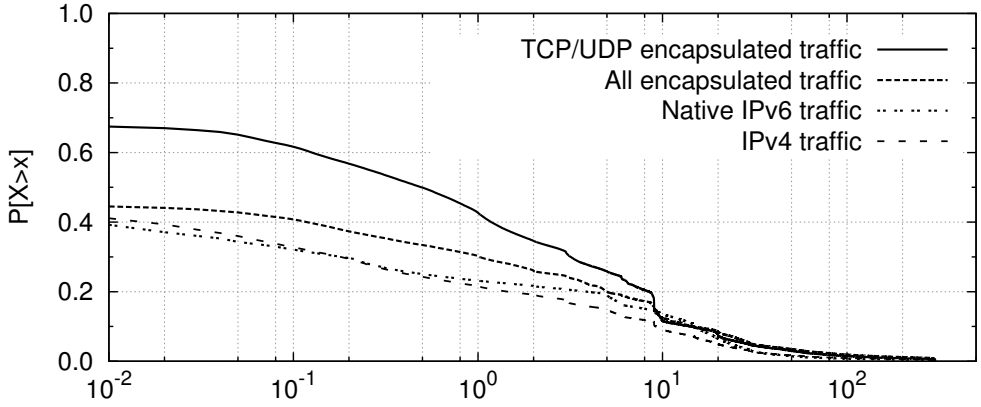


Figure 4.8: CCDF of Flow Duration.

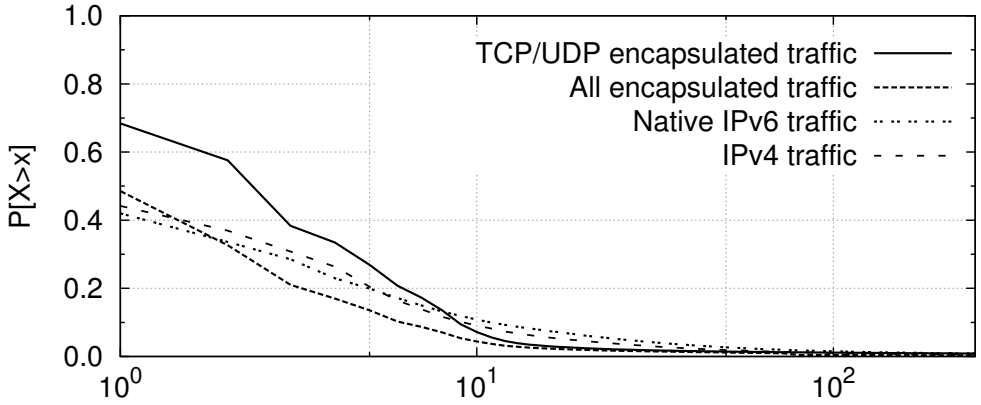


Figure 4.9: CCDF of Packets per Flow.

4.2.5 Duration and Size of Flows

The third group of characteristics is represented by flow duration, the number of packets per flow, and packet size (bytes per packet) statistics. For evaluation we employ *empirical complementary distribution function* (CCDF). We use the following formula to compute CCDF values:

$$\bar{F}(x) = P(X > x) = 1 - \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}\{x_i \leq x\} \tag{4.5}$$

where $\mathbf{1}\{x_i \leq x\}$ is the indicator whether the event $\{x_i \leq x\}$ has occurred or not. The CCDF function describes how often the selected variable is above a particular level. From all traces we filtered out four subsets of traffic: TCP or UDP encapsulated traffic (TCP/UDP), all encapsulated traffic (ALL), IPv6 native or decapsulated traffic (IPv6), and IPv4 traffic (IPv4). The subsets were chosen in order to compare the tunnelled traffic with other common traffic types. Further, for each of the subsets and each of the characteristics, CCDF has been computed. Figures 4.8, 4.9, and 4.10 show the calculated CCDFs.

The majority of the flow duration of all subsets (Figure 4.8) accounts for durations shorter than 10 seconds. The flow durations longer than 10 seconds represent only 11 % or less. The TCP/UDP contains much fewer short duration flows (TCP/UDP: 32.16 % $\leq$ 0.01 sec.; ALL: 54.66 %; IPv4: 59.84 %) which is explained by the absence of connection-maintaining flows necessary for other subsets. We can observe slightly increased frequencies of the flow duration between 7 and 11 seconds especially at TCP/UDP and ALL subsets. Hence, the tunnelled traffic generally contains fewer short duration flows than IPv4 or IPv6 traffic.

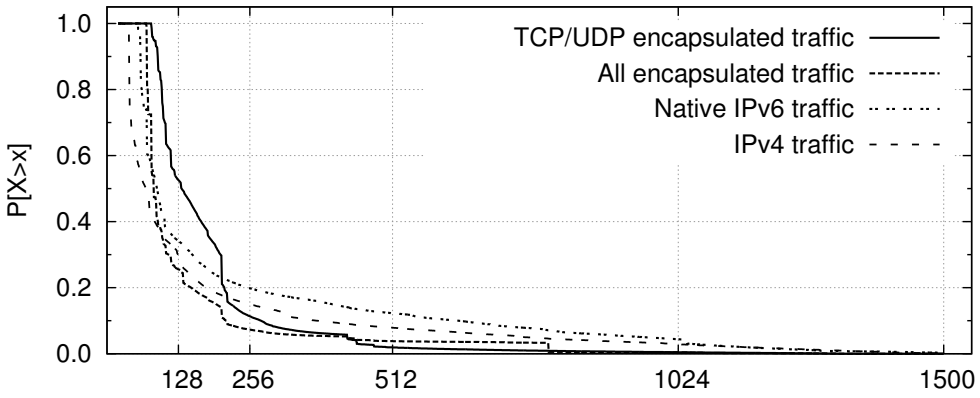


Figure 4.10: CCDF of Packet Size.

The packets per flow distribution (Figure 4.9) suggests that single packet flows form only 31.6 % in the case of TCP/UDP, 51.42 % in the case of ALL, 57.99 % and 55.82 % in other cases. We expected tunnelled traffic to behave in a similar way as IPv6 traffic. Nevertheless, we can observe a vertical shift between ALL and IPv6 CCDF. This shift is caused by single packet flows, and it is caused by DNS traffic to root DNS servers, which uses IPv6 protocol. The slope of CCDF for TCP/UDP and ALL is higher than the slope for other subsets, which indicates higher frequencies of certain packet counts. In conclusion, the distribution of the packet counts of the tunnelled traffic is slightly different from the distribution of the IPv4 and IPv6 traffic.

The last characteristic described by CCDF is packet size (Figure 4.10). In the case of encapsulated traffic, we consider the outer packet size including the encapsulation header. The earlier study of the packet size distribution mentions a significant difference between CCDFs of packet sizes. The authors of [155] state that the distribution of IPv4 traffic fits a heavy-tailed distribution, whereas IPv6 traffic does not. We expect the tunnelled traffic to have similar characteristics as IPv6 traffic, and thus the CCDFs are expected to be similar, too. The Figure 4.10 shows some discrepancies in this hypothesis. The packets larger than 400 bytes in the tunnelled traffic represent only 5 %, while in the IPv6 the portion is still 15 %. Furthermore, packets smaller than 70 bytes are not present in the tunnelled traffic, although they account for 26 % of IPv6. This is caused by a shift of graph to a higher packet size given by the encapsulation. The IPv4 header is usually 20 bytes long, and in case of Teredo, there are another 8 bytes for the UDP header. Even taking this shift into account there is still a noticeable drop at the size of 200 bytes which is caused by a high share of ICMPv6 and BitTorrent control traffic.

4.2.6 IPv6 Tunneled Traffic Analysis

In this section, we focus on characteristics of encapsulated IPv6 traffic for which we use the same dataset as in Section 4.2.4. We show HOP limit statistics, detected Teredo servers and geolocation characteristics. We discuss the most used TCP and UDP ports inside the tunnels.

Distribution of HOP Limits

Figure 4.11 shows HOP limit distribution for the encapsulated IPv6 traffic. The main difference from the TTL (Figure 4.4) and HOP limit (Figure 4.6) statistics is that the values here are distributed with much less entropy. The limits 21, 64, 128 and 255 are achieved and also the most frequent ones. This is caused by the fact that most of the traffic never traversed the IPv6 network and the HOP limit was therefore never decreased. In fact, when the values are lower, we can be reasonably certain that the packets already traversed the IPv6 network and are heading towards the IPv4 destination. The value 21 is used for Teredo bubbles by Windows 7 with Service Pack

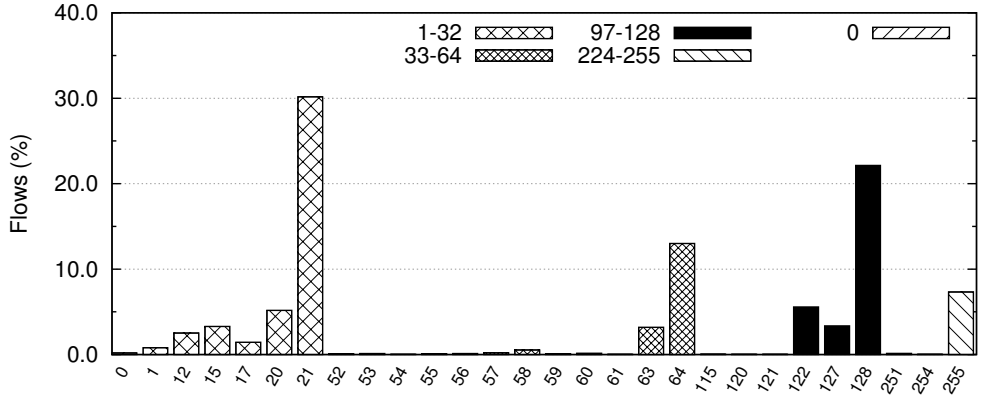


Figure 4.11: HOP Value Distribution of Encapsulated IPv6 Traffic.

1 and earlier. The Teredo bubbles are used as a special mechanism for NAT traversal, which is consistent with the fact that most of the clients are behind a NAT. We can see that some packets reach the value of the zero HOP limit, which is a known problem when the HOP limit is set as low as to 21. The value of 255 is used for IPv6 neighbour discovery messages so that when a host receives such packet with HOP limit lower than 255, the packet is considered invalid [156].

Teredo Servers

There are two ways of detecting Teredo servers. Firstly, we can look at the traffic using UDP protocol on port 3544, which is a well-known Teredo port, and select the addresses that communicate most often. The shortcoming of this approach is that some other services might be using the Teredo port and therefore the results might not be accurate. Since we are able to decapsulate Teredo traffic, we can derive IPv4 addresses of Teredo servers directly from Teredo IPv6 addresses. This way we can even detect Teredo servers that are not communicating directly throughout our observation points. Table 4.11a shows Top 10 servers that were discovered in the encapsulated IPv6 addresses. Using the WHOIS database, we confirmed that a majority of servers is operated by Microsoft, which is only to be expected since Teredo is a Microsoft technology. The most of Microsoft Teredo servers we identified are actually IP addresses of "teredo.ipv6.microsoft.com", which is the default Teredo server name configured under Windows. The address 83.170.6.76 has a hostname indicating that it serves as a Miredo server (Teredo implementation for Linux and BSD). The last address belongs to CZ.NIC (Czech top-level domain operator), which is known to promote IPv6 deployment in the Czech Republic and operates local Teredo and 6to4 servers.

Table 4.11b shows Teredo servers discovered as the most active on Teredo port 3544. This way we detect only Teredo servers that are establishing connections through our observation points. We can see that most users use Teredo servers in the United States or Great Britain to get IPv6 connectivity. This is known to increase the latency of such connections, and therefore we would recommend using local servers to Czech users, such as the CZ.NIC servers.

Location of Tunnel Endpoints

The geolocation statistics of tunnelled traffic are computed for Teredo and 6to4. We use encapsulated IPv6 addresses to determine the countries for each flow. Incoming and outgoing traffic is taken separately just as in Figure 4.7. The statistics are shown in Figure 4.12. The tunnelled traffic shows very different geolocation characteristics compared to native and decapsulated IPv6 traffic even though both are from the same link. Most of the native and decapsulated IPv6 com-

Server IP	Ratio	Owner	Country
65.55.158.118	28.33 %	Microsoft	US
94.245.121.253	27.98 %	Microsoft	GB
157.56.149.60	26.49 %	Microsoft	US
157.56.106.184	10.18 %	Microsoft	US
94.245.115.184	6.41 %	Microsoft	GB
83.170.6.76	0.04 %	B. Schmidt	DE
170.252.100.131	0.01 %	Accenture	US
94.245.127.72	0.01 %	Microsoft	GB
94.245.121.251	0.01 %	Microsoft	GB
217.31.202.10	0.01 %	CZ.NIC	CZ

(a) Based on Teredo IPv6 Addresses.

Server IP	Ratio	Owner	Country
94.245.121.253	43.24 %	Microsoft	GB
65.55.158.118	18.91 %	Microsoft	US
157.56.149.60	17.86 %	Microsoft	US
94.245.115.184	10.00 %	Microsoft	GB
157.56.106.184	6.50 %	Microsoft	US
94.245.121.254	0.72 %	Microsoft	GB
94.245.115.185	0.22 %	Microsoft	GB
65.55.158.119	0.18 %	Microsoft	US
83.170.6.76	0.18 %	B. Schmidt	DE
157.56.149.61	0.17 %	Microsoft	US

(b) Based on UDP Port 3544

Table 4.11: Top 10 Discovered Teredo Servers.

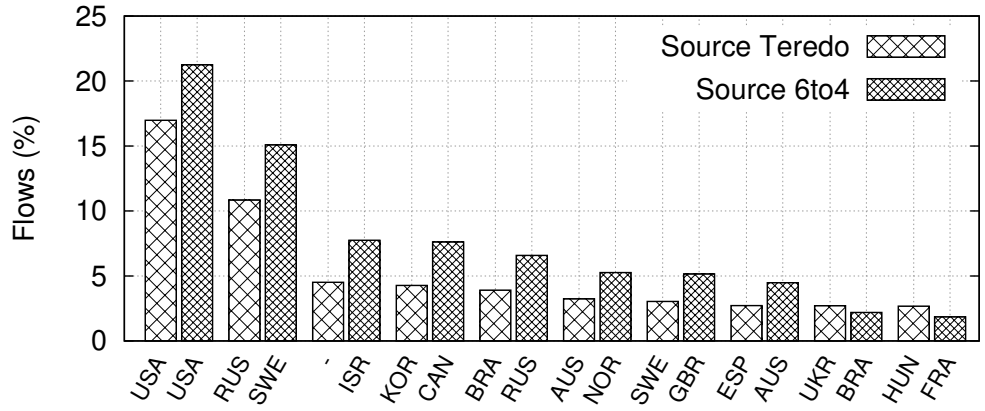
munication takes place inside the EU, while a large portion of tunnelled traffic communication is performed with the USA and Russia.

To identify applications that are using IPv6 connection provided by transition mechanisms, we created a list of the most used encapsulated TCP and UDP ports. We observed several ports that can be found both in the source and destination port Top 10. The source and destination ports Top 10 represent 32.0 % and 40.5 % of the traffic respectively. The well-known ports are - *HTTP - 80* (0.85 % of traffic as source port, 5.61 % as destination port), *HTTPS - 443* (0.58 % and 1.48 %) and *DNS - 53* (1.49 % and 1.48 %). Among the most frequent ports are ports 49001 (15.96 % and 9.91 %) and 51413 (10.12 % and 16.07 %) which are used by BitTorrent clients (namely Vuze and Transmission). We discovered that these ports are heavily used within the 6to4 tunnel as mentioned in Section 4.2.4.

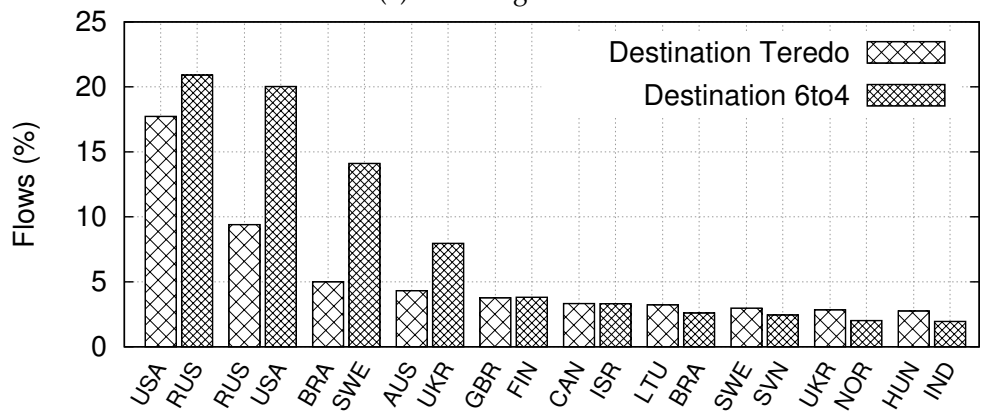
4.2.7 Evaluation of IPv6 Adoption

In this section, we describe the deployment of the IPv6 protocol with respect to tunnelled traffic. The overall statistics of IPv6 and tunnelled traffic are mentioned. We provide a historical comparison to our previous measurement [151].

The network activity shows the correlation with human activity. Both IPv6 and tunnelled traffic are considerably smaller during the weekend than during weekdays. As for the IPv6 traffic, the increase of traffic volume starts at 6 AM, reaches the peak around 11 AM and holds the high level till 4 PM. Then the traffic steadily decreases and reaches the minimum at 3 AM the next day. The tunnelled traffic shows a slow increase that starts at 6 AM and peaks around 6 PM. The decrease begins at 10 PM and reaches the minimum at 5 AM the next day. The possible



(a) Incoming Traffic.



(b) Outgoing Traffic.

Figure 4.12: Top 10 Country Distribution for Encapsulated IPv6 Addresses.

cause of this shift from the IPv6 diurnal pattern is the fact that the tunnelled traffic is widely made by BitTorrent clients.

We measured the tunnelled traffic back in 2010 on three CESNET border links to SANET, PIONIER and NIX. We found that the tunnelled IPv6 was responsible for 1.5 % of total flows, which is the same share as we measured today. However, the relative amount of bytes transferred has almost doubled from 0.66 % to 1.28 % of total bytes today. The share of the native and decapsulated IPv6 was only 0.10 % (0.21 % of bytes) compared to 3.39 % (4.42 % of bytes) today. The known services by port (HTTP, HTTPS and DNS) had a share less than 1 % of total flows. Today's share of these services is significantly higher (see Section 4.2.6). The measurements show that the overall usage of both tunnelling IPv6 transition mechanisms and native IPv6 has been rising.

We distinguish between the encapsulated and decapsulated tunnelled traffic, as mentioned in Section 4.2.2. The decapsulated tunnelled traffic is included in the measured IPv6 traffic. When we filter out the decapsulated Teredo and 6to4 traffic, they account together for 5.91 % of the measured IPv6 traffic. Teredo traffic takes part of 83.13 % of the decapsulated tunnelled traffic and 6to4 16.87 %. Hence we should not consider all measured IPv6 traffic as native IPv6 traffic.

The main contributor to tunnelled traffic was Teredo with an occurrence of nearly 89 % followed by 6to4 with over 11 %; therefore the relative amount of 6to4 traffic has increased. We have detected the use of 13 Teredo servers in the previous study compared to 53 today.

4.2.8 Conclusion

In this section, we have taken a detailed look at the IPv6 transition mechanisms. We have provided an improved version of our tool for investigating IPv6 tunnelled traffic. Considerable progress has been made with regard to understanding tunnelled traffic behaviour, especially concerning Teredo and 6to4 traffic. The results of this section suggest that encapsulated traffic differs from IPv4 and IPv6 in several characteristics including TTL values, geolocation aspect and flow duration. Moreover, we have provided the list of Teredo servers and described the evolution of IPv6 adoption.

4.3 Large-Scale Geolocation for NetFlow

Geolocation of network traffic is the process of identifying the geographical location of hosts using their IP addresses. The addition of this information to network traffic is useful for many activities, such as business advertisements, fraud detection and access control. Several types of geolocation can be identified. Active geolocation estimates the location of hosts mostly based on delay measurements [157, 158]. This results in high accuracy but comes at the expense of lack of scalability, and high measurement overhead [159]. On the other hand, passive geolocation relies on static datasets, such as databases, containing the geolocation information per IP address block. Online databases, such as geoPlugin [160], can be accessed easily from Web applications and often apply rate or request limiting. This makes them less suitable for bulk geolocation and high-interaction applications. Offline databases, such as MaxMind GeoLite [161] and IP2Location [162], do not have these limitations. Both the fact that databases can become outdated and that the majority of data used by passive geolocation approaches refers only to a few popular countries, impact the accuracy of these approaches [159].

Existing geolocation approaches are designed for on-demand, mostly small-scale purposes, where the geolocation is performed by analysis applications that retrieve the flow data from collectors. However, when geolocation needs to be performed in large, high-speed networks, these approaches are not scalable anymore. The main reason for this is that the dataset to be geolocated is too large for these applications, mainly because multiple flow exporters are typically exporting flow data to a single collector. Also, these applications are often developed for data visualisation, rather than bulk processing.

The goal of this section is to demonstrate how flow-based geolocation can be performed in a large-scale fashion. As a first step, we propose a prototype for exporter-based geolocation, which adds geolocation data to flow records before they are sent to a collector, which is an approach used in the previous section. As such, the actual geolocation can be distributed over multiple exporters instead of being deployed on a single collector. Since data analysis is almost always done at a collector or by analysis applications that retrieve data from the collector, we also propose an extension to the state-of-the-art flow collection and analysis tool *NfSen* [73] that adds native geolocation support. We define *native geolocation support* as the ability to process geolocated flow data in the same way as non-geolocated flow data with respect to storage, filtering, aggregation, and statistics generation. After presenting the two prototypes, we analyse the performance footprint of our approaches on the primary tasks of flow exporters and collectors and present several use cases that demonstrate the practical applicability of large-scale geolocation.

The remainder of this section is structured as follows. Subsection 4.3.1 provides an overview of related works in the field of flow-based geolocation. The main contribution of this work is described in Subsections 4.3.2 and 4.3.3, where we present how we perform large-scale geolocation on flow exporters and collectors, respectively. In Subsection 4.3.4 we describe the deployment of our prototypes and in Subsection 4.3.5 we present several use cases, which demonstrate their viability. Finally, we close this section in Subsection 4.3.6, where we draw our conclusions.

4.3.1 Related Work

Due to the increasing number of application areas for geolocation, many related works have been developed in the past. Closest to our work is *nProbe*, a NetFlow/IPFIX probe for exporting NetFlow data to a flow collector [163]. This flow exporter uses MaxMind GeoLite [161] for the conversion of IP addresses to geographical locations and AS numbers. The fact that geolocation by *nProbe* is exporter-based makes it scalable for use in large-scale networks. However, by storing the geolocation information in special fields, software support by collectors and applications is required to access this data. Geolocation by *nProbe* is therefore not transparent to any flow collector, which makes this approach different from our work.

Other related works, such as *SiLK* [164], *ntop* [165], and *Argus* [166], are collector-based. *SiLK* and *ntop* add geolocation information to flow data both in an on-demand fashion and as a post-processing step. As such, geolocation information is used purely for visualisation and native geolocation is not supported. *Argus* is more advanced, since it creates an extended dataset from the NetFlow data, in which it can also store geolocation data. As such, full geolocation support is provided as long as the geolocation data has been added as a post-processing step to the dataset before.

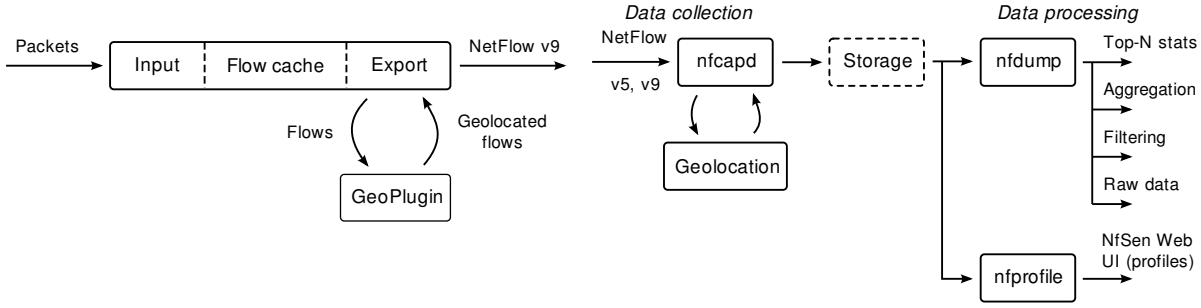
Besides exporter- and collector-based geolocation approaches, dedicated analysis applications have also been developed. These applications retrieve flow data from flow collectors and subsequently perform the geolocation. *SURFmap* [167, 168], a plugin for the flow collector *NfSen*, uses geolocation to visualise network traffic on a map using the Google Maps API. Due to a dependency on the Google Maps Geocoder for translating location names to coordinates, only a limited number of queries can be executed per day. A *geofilter* has been included since version 2.3, which aims to filter flow data based on country, region or city keywords as a post-processing step. The specification of this *geofilter* is one of the core elements of our collector-based geolocation prototype. Another application is *HAPviewer* [169], which is able to provide flow-level statistics per country of communication partners. Both *SURFmap* and *HAPviewer* perform geolocation on-demand and do not provide native geolocation support.

4.3.2 Exporter-Based Geolocation

Flow exporters can perform more tasks than only flow export nowadays, due to their increasing performance. We propose a prototype for exporter-based geolocation, which adds geolocation data to flow records in a way that is transparent to standard flow collectors. This means that the geolocation data can be accessed by any standards-compliant collector. Usually, multiple flow exporters send their data to a single flow collector. An exporter-based approach, therefore, aims to distribute the geolocation process over multiple devices. This improves the overall scalability in large networks and reduces the performance footprint on flow collectors.

Our exporter-based geolocation prototype has been developed as a plugin for the FlowMon [99] platform. Plugins can be used to alter flow creation, processing, filtering, and export. The architecture of FlowMon is shown in Figure 4.13a. Packets on the line are received by *input* plugins that store newly created flow records in the *flow cache*. As soon as a record in the cache has been expired (*e.g.* due to a timeout), it is removed from the cache, after which the *export* plugin takes care of placing it in NetFlow packets. Our geolocation plugin is depicted as *GeoPlugin*, which is executed by the *export* plugin just before the record is exported. *GeoPlugin* will do the actual geolocation and add the resulting data to the flow record.

To make sure that our exporter-based geolocation is transparent to any flow collector, NetFlow v9 has been chosen as the export protocol. It provides a fixed set of fields that can be used to store information about a flow. To add country-level information to flow records, we either have to use reserved (vendor proprietary) fields, or reuse some of the existing fields. Only country-level geolocation information is considered in this work, due to the poor accuracy of



(a) Exporter-Based Geolocation (b) Collector-Based Geolocation

Figure 4.13: Prototype Architectures for Large-Scale Geolocation.

geolocation databases with respect to region- and especially city-level geolocation data [159]. NetFlow's reserved fields are typically not supported by collectors, which leaves the reuse of existing fields as the only option to ensure transparency, compatibility, and early deployment. Since the *SRC_AS* and *DST_AS* fields are rarely used in typical flow monitoring setups (due to the lack of BGP data integration), we use these fields for storing the source and destination countries of IP addresses in flow records. After retrieving the geolocation data from the MaxMind GeoLite database, the resulting country-code is converted to a numerical value and stored in these fields. When flow collectors and analysis applications need to access the country-level information in the flow records, the *SRC_AS* and *DST_AS* fields need to be parsed and the values need to be translated to country codes.

There are many advantages of using the IPFIX protocol for exporting geolocation information. IPFIX is more flexible than NetFlow v9, supports more fields (named *IPFIX Information Elements*), and makes it easier to define enterprise-specific fields. Several fields for adding location information to IPFIX have been proposed already in [170]. However, those fields are only used for storing the location of an IPFIX flow exporter and are therefore not suitable for flow geolocation. Besides the flexible definition of fields, genuine IPFIX collectors are able to receive new fields and should be flexible enough to process them. However, no suitable IPFIX collectors were available at the time of writing and reusing NetFlow v9 fields served well for our purposes, even though this approach is highly discouraged. Note that switching to the IPFIX protocols is only a matter of adjusting flow exporter configuration.

4.3.3 Collector-Based Geolocation

Flow collectors typically aggregate flow data from multiple flow exporters, which makes them a suitable location to perform data analysis. *NfSen* is a popular collector and analysis tool, used by many network administrators in large-scale networks where performance and stability are essential. *nfdump*, an analysis tool included in *NfSen* that takes care of the actual data analysis, uses a flat-file database with an extensible format[1] to read flow data. Several extensions have been included before, such as SNMP interfaces, AS numbers, MAC addresses, and VLANs. We propose a new extension for storing source and destination country codes (based on ISO 3166-1) for IP addresses in flow records. For the same reason as explained for our exporter-based approach, only country-level information is considered. Besides the database extensions, also support for filtering, aggregating, and statistics generation based on the geolocation data need to be added to *nfdump*, to provide native geolocation support.

Besides *nfdump*, *NfSen* includes several other tools that need to be modified to provide native geolocation support in a flow collector. Among them is *nfcapd*, which receives flow data from flow exporters, performs the actual geolocation (using MaxMind GeoLite) and writes the data

1. The extensible format is supported by *nfdump* since version 1.5.7.

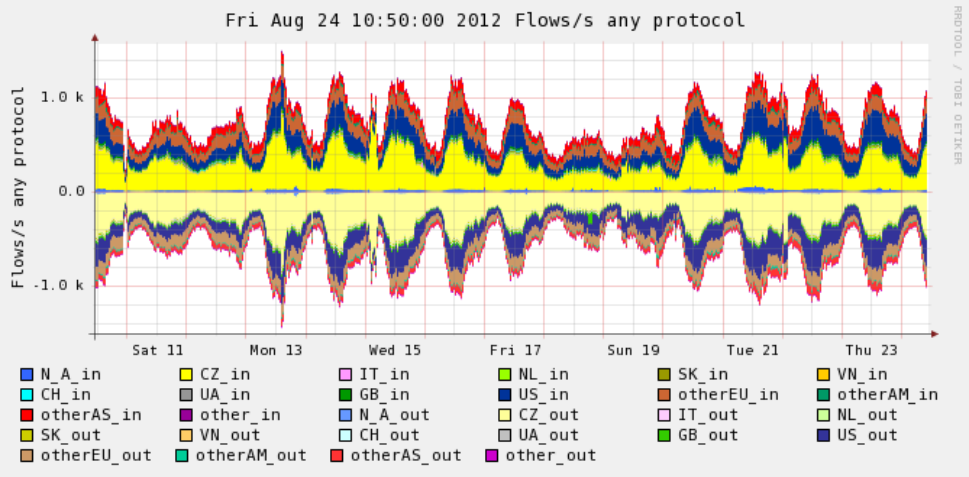


Figure 4.14: Screenshot of Collector-Based Geolocation Prototype.

to the disk. Obviously, *nfcapd* also needs to support the new flat-file database format of *nfdump*. Other modifications need to be made to *nfprofile*, which performs the traffic profiling for *NfSen*. The overall architecture of our collector-based geolocation prototype and the data flow between the various components is shown in Figure 4.13b.

A screenshot of our prototype is shown in Figure 4.14, which demonstrates one aspect of the native geolocation support: The traffic is now automatically profiled by country names. The introduced geolocation support is completely transparent to and compatible with other parts of *NfSen*, and provides near real-time, long-term traffic profiling based on geolocation. No additional tools are required. There are ongoing discussions with Peter Haag, the developer of *NfSen* and *nfdump*, about the integration of our geolocation modifications in the main source tree of these tools.

4.3.4 Prototype Deployment

The two previous sections have discussed our approaches to exporter- and collector-based geolocation. Both approaches aim to translate IP addresses to geographical locations in a scalable manner, for deployment in large-scale networks. To validate whether this aim is fulfilled, we have deployed both prototypes on the 10 Gbps Internet connection of the Masaryk University (CZ), which connects the campus network to the Czech national research and education network CESNET.

The primary tasks of flow exporters and collectors are data export and collection, respectively. Although geolocation adds a useful new dimension to this data, it should never interfere with the primary tasks of these devices. To analyse the performance footprint of our geolocation, we have measured the number of geolocation queries that could be performed per second. MaxMind GeoLite provides four data retrieval types, namely one file system-based (*standard*) and three memory-based (*memory cache*, *check cache* and *MMAP cache*) ones. Besides them, both IPv4 and IPv6 address geolocation have been tested, since they use different databases with different schemas. The results of the measurement[2] are shown in Figure 4.15 and reveal a clear performance increase when either *memory cache* or *MMAP cache* is used: Up to $15.7 \cdot 10^7$ IPv4 and $5.4 \cdot 10^7$ IPv6 addresses could be geolocated per second. Since flow records consist of two IP addresses, roughly $7.8 \cdot 10^7$ IPv4 and $2.7 \cdot 10^7$ IPv6 flow records can be geolocated per second. However, geolocation using memory-based retrieval methods is CPU-intensive and consumes

2. We have used a machine with the following configuration: Intel Xeon E5410 CPU at 2.33 GHz, 12 GB RAM, SATA disk with 7200 RPM and Linux kernel 2.6.32 (64 bit).

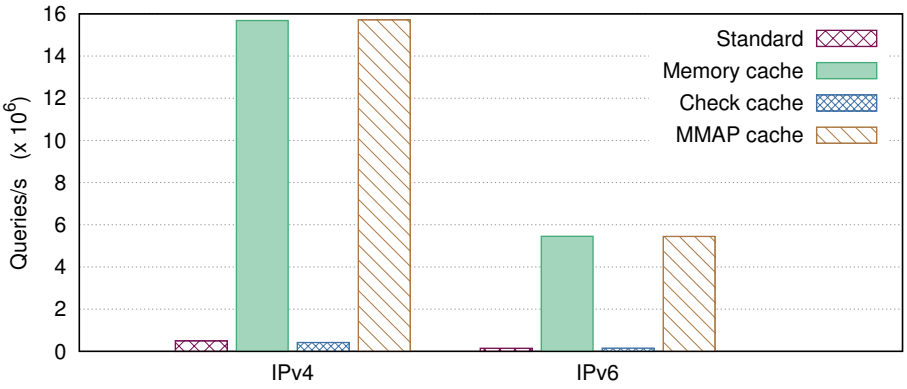


Figure 4.15: MaxMind GeoLite Country Database Performance.

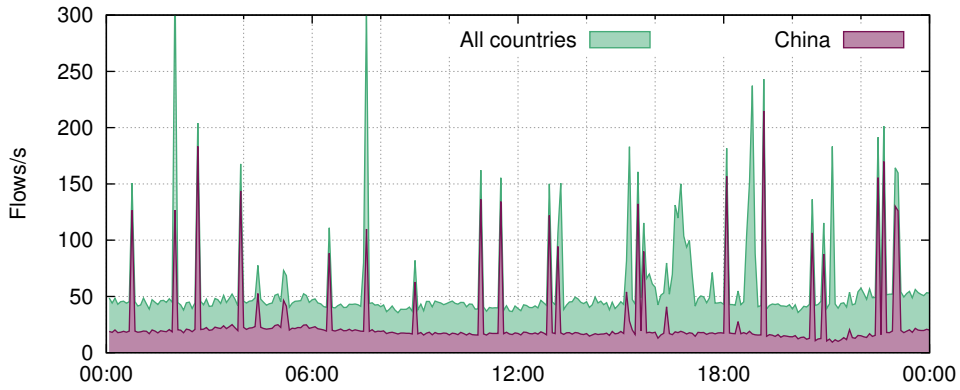


Figure 4.16: Incoming TCP SYN-Only Flows.

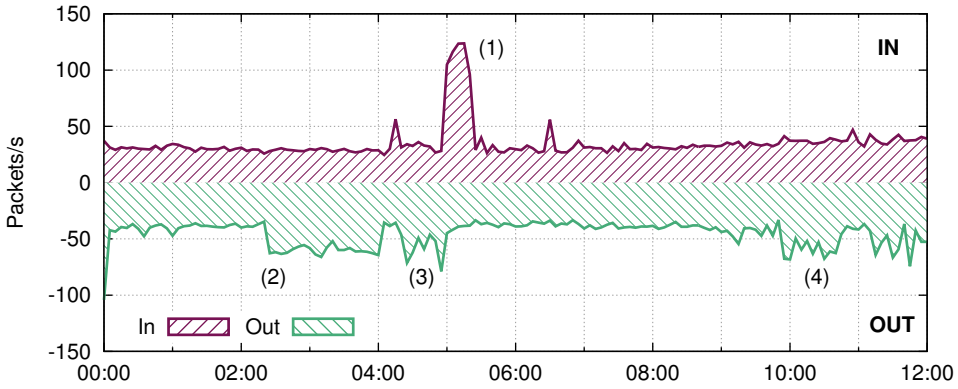
up to 100 % of the CPU time. The derived numbers for the geolocation performance can therefore never be reached in practice. In a theoretical case where either the flow export or collection process may consume 50 % of the CPU time, roughly $3.9 \cdot 10^7$ IPv4 and $1.4 \cdot 10^7$ IPv6 flow records can be geolocated per second. As a result, the performance of exporters and collectors in our deployment setup is not affected in any way, as we have up to $6.0 \cdot 10^3$ flow records per second to process. Neither will it on backbone links of CESNET, where up to $45 \cdot 10^3$ flow records are exported and collected per second.

4.3.5 Use Cases

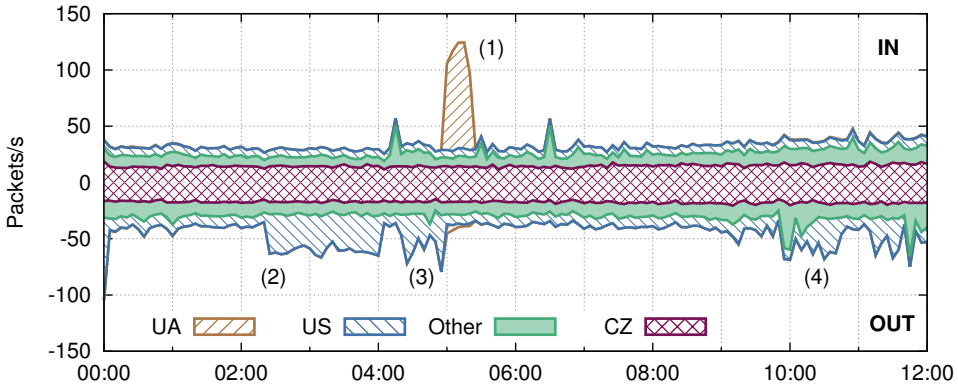
In this section, we present analysis results from our collector-based geolocation prototype, organised by two use cases. It is based on the deployment described in Subsection 4.3.4. The first use case demonstrates the applicability of our (pre-processed) geolocation approaches for the sake of anomaly detection. The second use case presents week-long traffic profiling at the country-level.

Anomaly Detection

It is a difficult task for network administrators to ensure security awareness in the daily barrage of scans, spamming hosts, zero-day attacks, and malicious network users, hidden in huge traffic volumes crossing the Internet. When security teams use geolocation for incident analysis, it is usually applied as a post-processing step. In contrast, our approaches perform geolocation as



(a) ICMP



(b) Geolocated ICMP

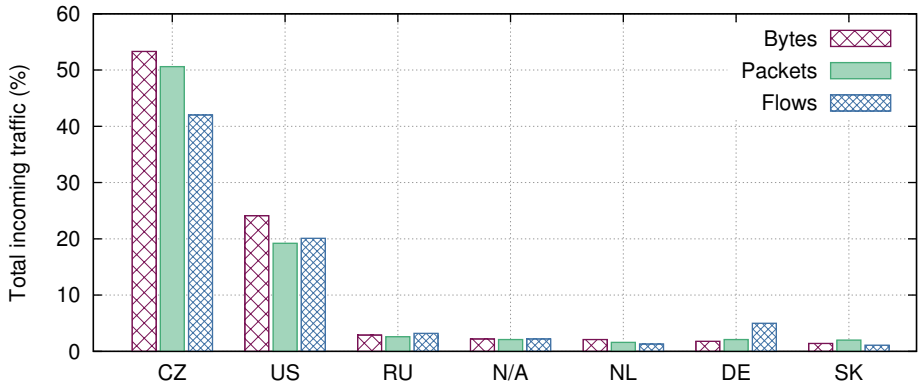
Figure 4.17: Geolocated and Non-Geolocated ICMP Traffic.

a pre-processing step, which allows using geolocation data in the detection process, among others.

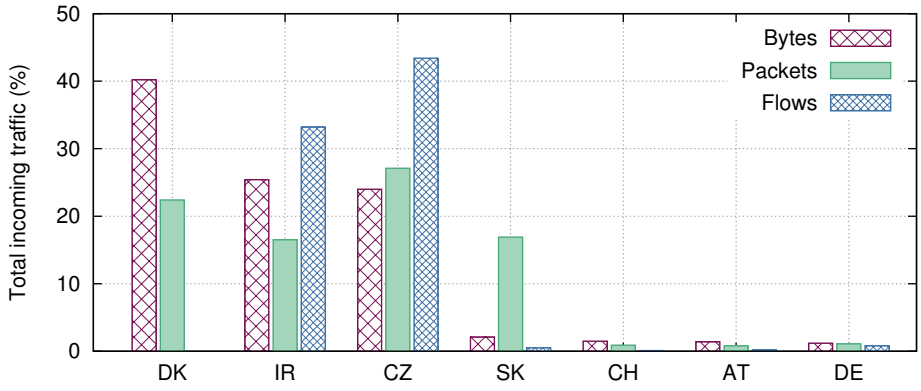
An example is shown in Figure 4.16, where the number of TCP SYN-only flows[3] per second during a period of one day is shown. These flows are typically created during the propagation phase of malware. We have identified that more than 40 % of them is originating from China. Also, the vast majority of traffic spikes is caused by Chinese connection attempts and only 22 % of Chinese TCP traffic completed the TCP-handshake. These findings demonstrate that this particular Chinese traffic is an interesting dataset for anomaly detection and malware analysis. Further investigation has revealed that using geolocation as a pre-processing step for anomaly detection can yield a dataset to be analysed that is only 40 % of its original size in this particular case, resulting in faster and less complex data analysis. This allows detecting anomalies that normally stay below the thresholds of anomaly detection systems. By creating traffic profiles for countries that are known to generate a higher than average amount of malicious traffic, such as China, Russia, Taiwan, Korea, and Thailand [171], we have been able to detect long-term stealth attacks.

Another example that demonstrates the advantages of native geolocation support in a flow collector is shown in Figure 4.17. Figure 4.17a shows an ordinary time-series of ICMP traffic for a period of twelve hours, while Figure 4.17b shows the same traffic but geolocated. Several anomalies - marked by numbers in the figures - can be identified, where the geolocated traffic makes the identification clearer due to the larger relative differences between anomalous and non-anomalous traffic. Peak 1 (incoming traffic) is caused by a Ukrainian host, scanning the

3. We denote TCP-flows that consist of a single packet with the SYN-flag set by *TCP SYN-only flows*.



(a) IPv4



(b) IPv6

Figure 4.18: Distribution of Incoming Traffic over Countries (1 Week).

complete university network using ICMP ECHO. Replying hosts were later contacted on TCP port 4899. Peaks 2, 3, and 4 (outgoing traffic) are caused by foreign hosts, which are sending spoofed UDP traffic to the university's DNS server to perform an amplification attack. This traffic is blocked by the firewall and ICMP Destination Unreachable is returned to a spoofed source IP address located in the United States.

When network administrators analyse anomalies using plots such as the one in Figure 4.17a, they can identify the peaks and start to filter the data to determine the hosts involved in an anomaly. However, plots similar to the one in Figure 4.17b make these tasks faster and simpler: Since the traffic is already pre-processed by country names, only the datasets related to a specific country need to be analysed. In the case of Peak 2, for example, this results in a dataset that is only half of its original size.

Traffic Profiling

Traffic profiling is the process of analysing the distribution of services and protocols in a network, such as the number of packets related to Web traffic or the number of flows of a certain type. When geolocation is applied, also statistics related to geographical locations can be generated, such as the number of connections to a certain country. Although this is also possible using existing geolocation approaches based on post-processing, our approach is able to generate these statistics in real-time and for the complete traffic mix. In this subsection, we provide two examples of geolocation-based traffic profiling: IPv4 vs IPv6 usage statistics, and HTTPS profiling.

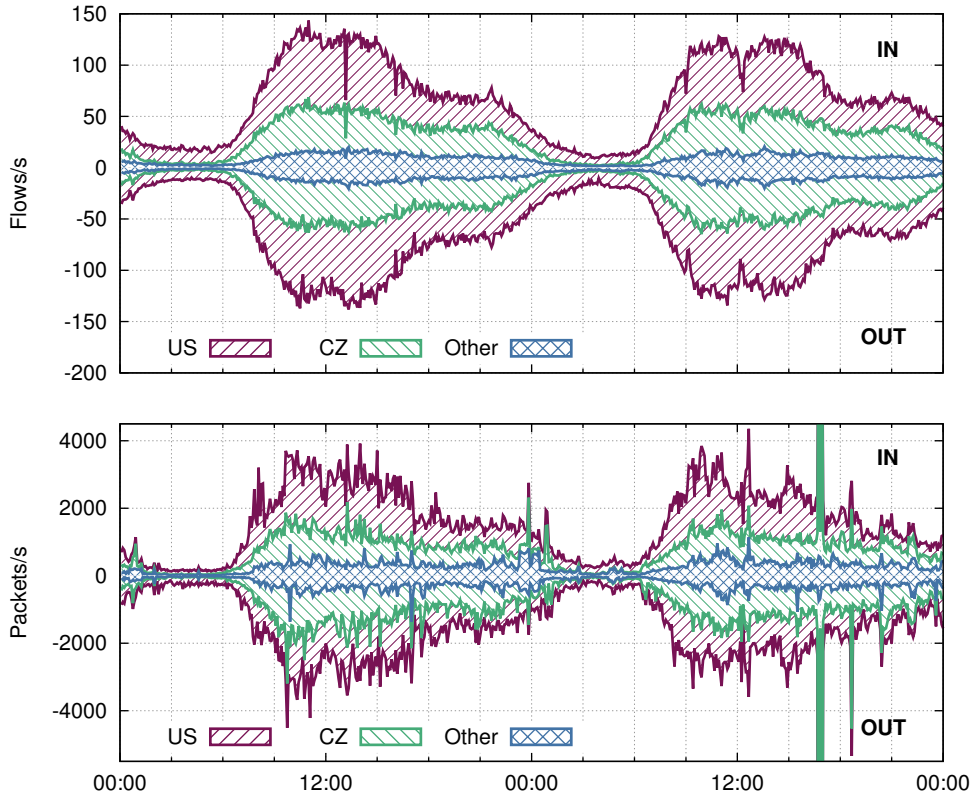


Figure 4.19: Distribution of HTTPS Traffic over Countries.

One method for measuring the worldwide spread of IPv4 and IPv6 deployment is based on counting the number of IPv4- and IPv6-enabled ASNs, respectively. Another method is to measure the percentage of IPv4 and IPv6 traffic per country. For our measurement point in the Czech Republic, the distribution of IPv4 traffic source countries is shown in Figure 4.18a. The fact that the United States is the source country generating the second most IPv4 traffic is not a surprise, given the fact that US-based social networks (Facebook, Twitter, LinkedIn) and content providers (Akamai, Google, Microsoft) generate a significant portion of the worldwide traffic [120]. This is confirmed by both manual inspection of the traffic and Figure 4.19, which shows that almost half of the HTTPS traffic, which is commonly used by those services, is going to and coming from the United States. Although the headquarters of these companies are all in the United States, their data are usually hosted closer to the service users, in regional data centres. This is achieved by using geolocation-aware-DNS (GeoDNS), which provides a means for service users to connect to the server that is closest to them from a geographical point of view. By probing the remote hosts in our datasets actively using ICMP ECHO, it is easily verified that the hosts are definitely not located in the United States, due to resulting non-transatlantic round-trip times. This is an example of a major inaccuracy of geolocation databases, which geolocate IP addresses to the companies' headquarters, instead of the real location of the hosts.

The distribution of source countries of IPv6 traffic is shown in Figure 4.18b. Next, to the Czech Republic, much traffic is received from Denmark and Ireland. Closer analysis has revealed that the Danish traffic is caused by large repository synchronisation (using FTP) by two data hosting providers. On the other hand, the Irish traffic is generated mainly by Google and Facebook. Surprisingly enough, this traffic is now geolocated to their real physical location. Google is known to have a large data centre in Ireland and some Facebook partners, such as Zynga, host their services in the Amazon data centre in Ireland as well.

4.3.6 Conclusions

In this section, we presented two prototypes for flow-based geolocation in large-scale networks. Existing approaches to flow-based geolocation have shown not to be suitable for deployment in those networks or to be non-transparent to flow collectors. By integrating geolocation into flow exporters and collectors, we include country-level information in flow data before the actual data analysis takes place. As such, geolocation is performed in real-time as a pre-processing step, instead of the usual post-processing. This allows network administrators to filter, aggregate, and generate statistics based on the geolocated data in a continuous fashion. Measurements have shown that the performance footprint of the geolocation process on the actual flow export and collection is negligible.

To demonstrate the applicability of real-time geolocation in large-scale networks, we have shown several examples by means of two use cases. For example, the use case on anomaly detection has shown that geolocation can help reducing the dataset to be analysed by analysis applications dramatically. If certain countries are expected to generate more malicious traffic than others, traffic from these countries can be pre-filtered. Analysis applications, such as anomaly detection systems, can then analyse the smaller dataset, which results in shorter detection times.

Implementation of the FlowMon geolocation plugin and *nfdump* geolocation patches are available at `http://ics.muni.cz/~celeda/geolocation/`.

4.4 Network Traffic Characterisation Using Flow-Based Statistics

A lot of research is being performed in the areas of network traffic classification, anomaly detection, and network security in general. Researchers involved in these areas often evaluate their methods using live network traffic. However, performing research on live network traffic includes several caveats. First, repeating experiments on live traffic is infeasible. Second, the traffic has to be thoroughly described, since most methods heavily depend on the properties of the observed traffic. Last but not least, the network traffic's properties can change during the research cycle, which can lead to suboptimal results. Although this work does not directly utilise application flow monitoring, it provides a method of ensuring repeatability of flow monitoring experiments, which is useful both for basic and application flow monitoring.

To be able to repeat the experiments, a packet trace must be captured. Considering the speed and utilisation of current uplinks of the research organisations, hundreds or even thousands of gigabytes of network data would have to be stored. Together with privacy issues, this makes sharing such datasets almost impossible.

Description of datasets is an important part of any work concerning network traffic. The properties of the traffic highly correlate with the results of any experiments. In case the researchers are not able to share their dataset, its description should allow other researchers to find a similar dataset which would exhibit similar results. Moreover, such a description can be checked for consistency thus ensuring that the results do not unduly fluctuate over time.

The goal of this work is to provide a simple method for discerning different types of network traffic. This method allows us to compare network traffic from live networks and even packet traces and determine which traffic samples show similar properties. Finding similar traffic is highly desirable as it allows the independent evaluation of published experiments. Moreover, when applying the method continuously to a specific network, changes in the properties of the network can be not only observed but also quantified.

Flow-based characteristics [15] are used by our method as they are easier to obtain than packet-based characteristics. Moore et al. [172] list a number of discriminators that can be used for classifying traffic. We select a small subset of discriminators obtained from NetFlow v5 [20] records that are available from most network devices. Our approach is generic, and the set of

discriminators can be extended for specific purposes. We show that the chosen discriminators provide enough information to distinguish between various networks.

We analyse the traffic of five campus networks within the Masaryk University. The networks have different properties as they contain a different number of servers, workstations, personal client stations, and portable devices. The networks and the methodology are described in detail so that other researchers can repeat our steps and evaluate their own network traffic.

We show that it is possible to distinguish between networks based on simple characteristics derived from flow information. Day-night and weekday patterns in the traffic are important phenomena that need to be taken into consideration when deriving characteristics. The characteristics used in this section do not show any significant correlation which indicates that all of them contribute to the description of the network traffic.

The rest of the section is structured as follows. Related work is surveyed in Subsection 4.4.1. We describe the methodology for characterising network traffic in Subsection 4.4.2. Results of applying the proposed method on several different networks are shown in Subsection 4.4.3. The section is concluded in Subsection 4.4.4.

4.4.1 Related Work

Statistical properties of Ethernet traffic were studied by Leland et al. [173]. They discovered that the traffic is statistically self-similar, which was later confirmed by several studies [174, 175]. These studies also showed that detailed characteristics, such as packet inter-arrival times, show large deviations and burstiness. Our study makes use of traffic flow properties which are aggregated over the whole network traffic and therefore much more stable.

Fraleigh et al. [176] performed packet-level traffic measurement on the Sprint IP backbone. They showed that the measurement interval affects peaks in packet rate and bit rate. Using flow-based analysis makes it possible to normalise the data. Otherwise, it would be more difficult to compare the results from different networks. The authors also showed that different networks have a different week and day-night patterns, as well as traffic types and packet size distribution. There was also a discrepancy between flows per second and bits per second for different networks. The authors argued that the network properties depend on the customer type of the link and its location.

Fomenkov et al. [177] studied network traffic behaviour on long-term data samples. The samples were captured from different networks, one to eight times a day, once per month for 60 to 120 seconds. Only packet headers were stored. The authors used packet, byte, and flow rates as well as a number of source-destination IP addresses to describe long-term changes in Internet traffic characteristics. The burstiness of the traffic in combination with short measurement intervals impeded the day-night and weekday patterns. Even the expected long-term growth of the traffic was not observed. This advocates using longer measurement intervals in our own work. Despite the intricate nature of the samples used, the authors reported that the average packet length increases with traffic growth. They also observed differences in the composition of traffic transport protocol usage between different networks.

Report of traffic properties measured on a single backbone network link was given by John and Tafvelin in [178]. They collected 148 traces of 20 minutes duration in a single month. The report focused on IP and TCP protocol usage, namely the distributions of IP transport protocols, IP protocol properties, IP and TCP options and flags. Authors also confirmed that the packet size distribution became bimodal as reported by some of the previous researches. A DoS attack was discovered in the collected sample, which caused an anomaly in the number of UDP packets in one of the samples. The report implies that knowing the composition of network traffic is important for the research of anomaly detection and network security.

Network	Packets	Bytes	Flows
Faculty of Informatics	227.1 G	236.4 T	3.6 G
Institute of Computer Science	107.3 G	106.2 T	0.7 G
University Campus Bohunice	449.8 G	473.9 T	4.1 G
Virtual Switching Segment	1 119.2 G	1 158.3 T	11.7 G
Masaryk University	1 366.6 G	1 427,7 T	20.1 G

Table 4.12: Measured Networks.

Benson et al. [179] described traffic characteristics of data centres. They performed a top-down analysis of ten data centres identifying applications used and their communication patterns. The authors observed flow-level and packet-level communication characteristics such as active flows per interval, the distribution of flow sizes and lengths, packet sizes and packet inter-arrival times. They noticed day-night and weekly patterns in the communication and that there is a difference between core and edge links as well as some of the data centres. Their analysis supports our belief that it should be possible to differentiate between network links based on traffic characteristics.

A characterisation of ISP traffic was performed by García-Dorado et al. [180]. The authors attempted to provide accurate, extensive, and quantitative measurements of application usage, bandwidth utilisation, and user preferences. They compared customers of different networks and access technologies for long periods of time. Unlike previous findings, the daily patterns are reported to be quite invariant, although the weekdays were different from weekends in a campus network. Application traffic shares differed significantly between the individual networks. Moreover, notable changes were observed on the same link during the measurement period. This supports our assumption that measurements on live networks should be well documented and validated.

All of the above-cited works focus on describing the properties and characteristics of network traffic. The key difference to our work is that we identify characteristics that vary between the networks and show that they can be used to differentiate and describe the networks in a uniform manner.

4.4.2 Methodology

To be able to describe network traffic and find key properties that allow us to discern different networks, we collected data from several different parts of our campus network. Understanding the purpose of each network is imperative for the correct interpretation of the data collected. Therefore we describe the networks used for collecting the data in detail. This section describes the processes and tools used to collect the data and the collected data itself.

A Description of the Networks

We continually collect data from individual networks on our campus. We chose five networks that are very different by their nature and utilisation. For each network, we selected two months of data, January and March. The networks are expected to exhibit different characteristics between the two months since there is an exam period at Masaryk University in January and March is a standard midterm month. A summary of networks and the collected data is presented in Table 4.12.

The **Faculty of Informatics** (FI) network represents a network of a university faculty. It connects staff offices, computer labs, and faculty servers. The faculty has its own Eduroam infras-

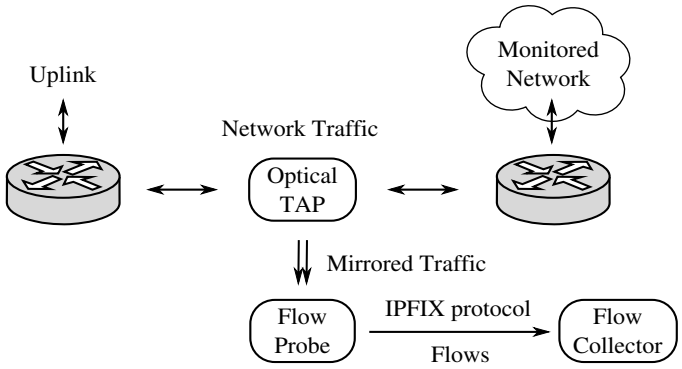


Figure 4.20: Flow Monitoring Architecture Schema.

tructure which can be observed in the network. The network also contains servers with the information system for the entire Masaryk University.

The **Institute of Computer Science** (ICS) has its own network connecting staff offices and a small server infrastructure to support office computers such as remote storage or update servers. Unlike the Faculty of Informatics, it does not connect computer labs.

University Campus Bohunice (UCB) is a large campus building with hundreds of offices, computer labs, and a large library. The faculty of Sports Studies and Faculty of Medicine are situated in the campus building. The Central European Institute of Technology (CEITEC) is also located on these premises, and it generates a large volume of data due to intensive scientific computing.

The **Virtual Switching Segment** (VSS) contains a server segment and also the Eduroam wireless network concentrator. Every Eduroam connection at the university goes through this network. Servers supporting the Masaryk University IT infrastructure, such as the Economic and Administrative Information System of Masaryk University or digital libraries, are also located in this network.

Masaryk University's (MU) network is measured as a whole at the uplink to ISP. The communication of every subnet belonging to the university is observed at this point with the exception of internal communication.

Data Collection

We have a flow probe [15] monitoring each of the networks in our campus. The data from the probes is sent to a flow collector where it is stored for further processing. The flow monitoring architecture is depicted in Figure 4.20.

Flow exporters are configured to create flows with an active timeout of 300 seconds and an inactive timeout of 60 seconds. The flows are terminated using only these timeouts, TCP FIN and RST flags are not used for the flow termination. The exporters use a standard flow key consisting of IP addresses, L4 protocol and its ports. It is important that exporters have the same configuration, since different configurations may result in a different number of exported flow records. As we will show, the number of flow records is an important characteristic of the network.

The flows from all probes are collected on a single collector. No sampling is used in the flow creation or collection process; therefore we have a complete flow trace of the measured network in a given period of time.

Type	Statistic
Basic Volume	Bytes per second Packets per second Flows per second
Advanced volume	Flow size in packets, bytes Flow connection length
Layer 3	Source host count Destination host count IPv4 / IPv6 bytes, packets, flows
Layer 4	IPv4 / IPv6 TCP bytes, packets, flows IPv4 / IPv6 UDP bytes, packets, flows IPv4 / IPv6 ICMP bytes, packets, flows IPv4 / IPv6 other protocol bytes, packets, flows
Layer 7	Top 10 ports by packets, bytes, flows

Table 4.13: Collected Statistics.

Data Preprocessing

This subsection describes how the raw flow data was processed. First, we generate statistics for short intervals so that the amount of data is reduced. These statistics do not carry any personal information; therefore they can be shared with other researchers without privacy concerns. Then we process these statistics, look at daily and weekly patterns, their averages and deviations. We identify characteristics which differ between the networks, describe them and use them to differentiate between the networks. To be able to compare the traffic characteristics of different networks, we need to normalise the volumetric properties of the traffic.

Individual flow records are not needed to describe network traffic. Instead, the records must be aggregated, and statistical indicators must be derived from the raw data. Since most flow-based frameworks process data in 5-minute intervals [73], we chose the same interval for generating aggregated statistics.

We generate five types of statistics from the original flow records: *Basic volume*, *Advanced volume*, *Layer 3*, *Layer 4*, and *Layer 7* statistics. These statistics are listed in Table 4.13. There are 41 collected statistics in total when the bytes, packets, and flows statistics are counted separately.

The Advanced volume statistics represent the detailed volumetric characteristics of the network. Therefore, we compute mean, variance, standard deviation, median, and all percentiles from 0.55 to 0.95 with an increment of 0.05. Percentiles lower than the median are superfluous since half of the flows in every network are composed of single packet flows, which amount to zero length flows with small packet sizes (typically 40 bytes). We also compute covariance and correlation for each pair of these statistics. The port statistics are stored as a table of top 10 ports sorted by either packets, bytes, or flows. We also the compute overall top port statistic for our analysis.

4.4.3 Results

We carefully considered all the acquired data and divided it into seven areas of network behaviour that are studied in more detail. This section describes our findings and decisions taken when analysing the collected data.

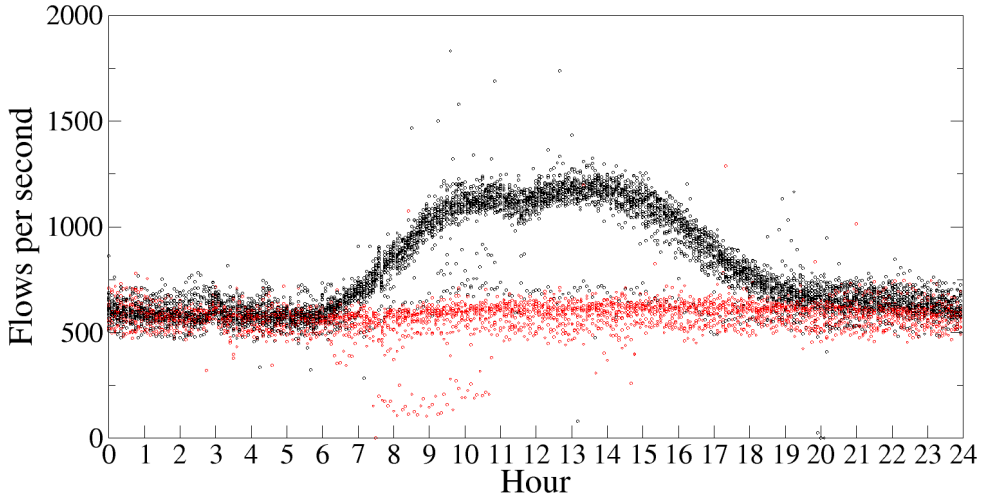


Figure 4.21: Flows per Second in the UCB Network.

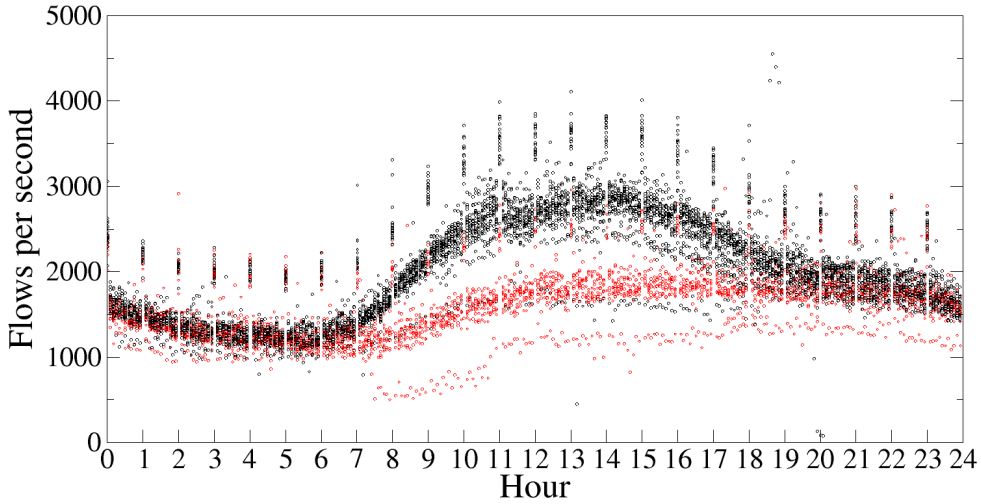


Figure 4.22: Flows per Second in the VSS Network.

Day-Night Pattern

The day-night pattern is a key element of change in every network with human users. It is evident that we cannot compare traffic captured during the day with traffic captured at night. The reason is that the traffic pattern during the night generally contains much less user-generated traffic and the properties are very different, as we present further in our analysis. The day-night pattern is best observed in flows per second since it expresses the number of connections. Using bytes or packets per second distorts the pattern since there are heavier flows during the day than at the night [181]. It is possible to use the number of hosts to show the day-night pattern, but the results are very strongly correlated; therefore we decided to use flows per second.

The day-night behaviour of the network discloses a lot of information about the purpose of the network. Figure 4.21 shows flows per second in University Campus Bohunice network in January. The data from all the days is superimposed in the graph so that the day-night pattern is clearly visible. The red colour shows the weekends, and the black is for workdays. It is possible to even see a lunch break in the data at half past eleven. The weekends are flat and do not exhibit the day-night patterns since there are not many users at the weekend.

Network	Day	Night	Day-night Ratio		Week Ratio	
			January	March	January	March
UCB	13	5	1.96	2.08	1.85	1.87
ICS	13	1	2.25	1.70	2.24	1.52
MU	14	5	3.08	3.62	1.45	1.95
FI	10	5	2.04	2.16	1.47	1.64
VSS	14	5	2.16	2.73	1.43	1.77

Table 4.14: Day to Night and Workday to Weekend Ratio According to Flows per Second.

Figure 4.22 shows the Virtual Switching Segment network which has quite different properties. Since there are Eduroam users and servers that are accessed from outside the university, we can see that there is a day-night pattern visible even on the weekends. Moreover, we can see periodical communication which is caused by automated server backups and updates.

For each network, the most and the least busy hour of the day according to flows per second was determined (as shown in Table 4.14). Given the property of the network, the *day-night ratio* will be, from now on, computed as a ratio between the average of the property during the busiest hour in the day and the least busy hour at night. Using day-night ratios specific to each network allows us to compare the behaviour of each network regardless of the absolute size of the network.

Table 4.14 also describes the day-night pattern for all networks. We processed the data from January and March 2015 separately to show the difference between the exam period and the teaching period. The day-night ratio expresses the ratio of flow count as described earlier. The largest difference between the day-night ratios in January and March is in the traffic from the entire university and the VSS traffic. This is a seasonal change caused by students leaving the campus and returning only for their exams. The traffic from faculty networks (FI, VSS) increases only slightly in the teaching period, indicating that the faculty networks are not highly utilised by students. The ICS network shows the opposite trend, which is likely caused by project deadlines and people working harder after the Christmas vacation to meet them.

We also computed an interval which has the maximal ratio of average flows per second to the rest of the day. We have observed two types of network. The activity in networks FI, MU, and VSS start at circa 7:00 and ends at 23:00, while the activity in networks UCB and ICS start at 8:00 and ends at circa 17:00. This can be attributed to the fact that UCB and ICS networks are used while people work, and the rest of the networks are used while people are awake.

The day-night pattern provides important information about the networks and their usage. We can use figures from Table 4.14 to describe the day-night pattern of these networks.

Weekday Pattern

Another significant pattern that can be observed in network traffic is the weekday pattern. Figures 4.23 and 4.24 show the weekday pattern in January for UCB and VSS. The graphs are constructed by superimposing the data from individual weeks. There are significant differences between the networks. The weekend traffic at UCB is almost flat, but VSS shows behaviour that suggests that there is a significant number of users communicating over the network even at weekends.

The difference between work days and weekends can be expressed as a ratio of average flows per second measured during the busiest hours on workdays and at weekends. Table 4.14 shows the results for January and March. This characteristic shows a correlation with the day-night pattern. We presume that if the network is accessed by other users, such as students, outside

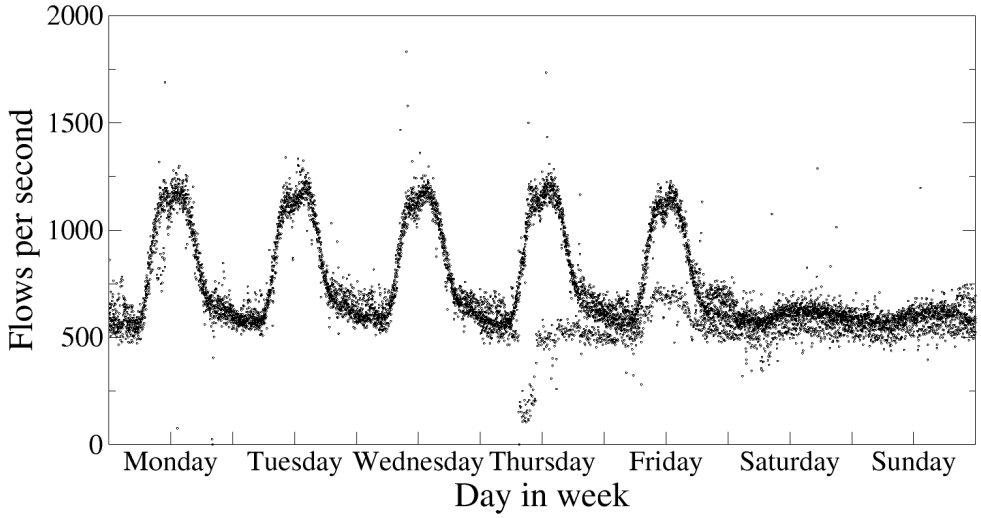


Figure 4.23: Flows per Second in the UCB Network.

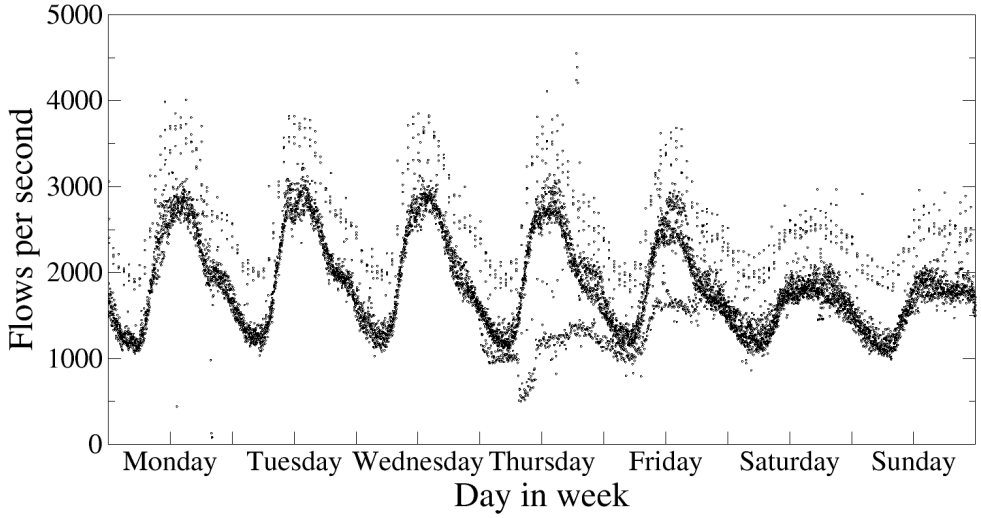


Figure 4.24: Flows per Second in the VSS Network.

working hours, it will probably be accessed also on weekends by the same users. However, we can still employ this statistic to differentiate between networks since the correlation is not strong and the statistic still adds new information.

Flow Characteristics

Using basic network characteristics such as packets per flow, bytes per packet, flow length, or a number of hosts is difficult due to day-night and weekly patterns. The variance of these variables renders averaging the values useless. Therefore, we have taken another approach. Average values are computed for whole weeks, and the averages are used as base values. The variance of week averages computed over the months is on average less than 10 %, which is low enough to use these values to characterise the networks.

Bytes per packet statistic had little variance over the measured networks, and it gives almost no information to discern the networks. We found that flow length and packet per flow statistics differ more significantly, as shown in Table 4.15, and can be used to discern the networks. The day-night ratios show that the networks with similar averages might have different day-

Network	Length of The Flow		Packets per Flow	
	Week avg.	Day-night rat.	Week avg.	Day-night rat.
UCB	10.07 s	2.18	112.86	1.28
ICS	10.34 s	1.68	205.36	0.66
MU	13.09 s	2.13	71.79	0.81
FI	5.40 s	1.77	67.60	0.70
VSS	7.14 s	2.04	106.25	0.94

Table 4.15: Average Length of Flow and Packets per Flow.

Network	Source Hosts		Destination Hosts	
	January	March	January	March
UCB	2.66	2.51	1.91	1.90
ICS	2.76	3.11	1.99	2.35
MU	2.49	2.66	1.65	2.08
FI	1.24	1.35	1.22	1.31
VSS	2.93	4.93	2.48	4.12

Table 4.16: Source and Destination Hosts Day-Night Ratios.

night behaviour for the same statistics. Therefore, the ratios add new information to the weekly averages.

The average number of hosts is a statistic directly related to the size of the network and cannot be used to describe the type of the network without normalisation. Therefore, we use only the day-night host ratios, as shown in Table 4.16, which are independent of the absolute number of hosts. Source and destination directions are determined by the direction of the flow. There is a significant difference between host ratios in January and March for the VSS network. The UCB is quite indifferent in comparison, as the ratios remain almost the same. This shows that the statistic can be used to find differences between the networks.

We have shown that comparing basic characteristics on networks must be done in sufficiently large intervals that are the same for all networks. In our case, the average values taken over an entire week proved to be stable enough to be used as a network characteristic. The ratio between characteristics measured during the day and night period also provides relevant information about the behaviour of the network and can also be used as an important network characteristic.

IPv6 Utilisation

The utilisation of the IPv6 protocol in the network is a Layer 3 characteristic that indicates the technological readiness of the network. The adoption of IPv6 is slow and IPv4 traffic is still dominant. We use this fact to differentiate between networks with different levels of IPv6 readiness. The amount of IPv6 traffic is affected by IPv6 ready servers in the networks and IPv6 services accessed by users outside the network. Table 4.17 shows the ratio of IPv6 to total traffic in flows, packets and bytes. The packets and bytes ratios are strongly correlated; however, they differ from the flows statistic. The UCB, ICS, and MU networks clearly stand out in these characteristics.

Protocol Share

We investigated the differences in Layer 4 protocol usage in the datasets. TCP and UDP are the dominating protocols in all networks, and the shares of ICMP and other protocols are so

Network	Flows	Packets	Bytes
UCB	0.02 %	0.01 %	0.00 %
ICS	5.58 %	12.35 %	13.57 %
MU	12.98 %	1.94 %	1.41 %
FI	3.22 %	2.04 %	2.10 %
VSS	4.66 %	0.23 %	0.17 %

Table 4.17: IPv6 Utilisation.

Network	Day		Night	
	Tcp	Udp	Tcp	Udp
UCB	38.52 %	59.76 %	19.44 %	77.66 %
ICS	41.26 %	56.88 %	28.20 %	68.84 %
MU	55.55 %	43.03 %	42.99 %	54.25 %
FI	49.96 %	49.07 %	25.15 %	73.49 %
VSS	30.67 %	67.76 %	19.30 %	78.00 %

Table 4.18: TCP and UDP Share in Day and Night in Flows.

negligible that they can hardly be used to describe the networks. The shares of TCP and UDP differ significantly between day and night time. Table 4.18 shows the average TCP and UDP shares during the day and night in flows.

The increase in TCP traffic during the day is caused by users since most of the services use the TCP protocol. We also investigated protocol shares in packets and bytes and found that they match the overall packet to flow and byte per packet ratio. The results show that the TCP to UDP ratio differs between the networks and can be used for their description.

Most Frequent Ports

Traffic analysis by port numbers provides evidence of application usage in the network. Although the port numbers are not considered to be accurate enough for application identification [182], most of the traffic adheres to well-known port numbers, which is accurate enough for our purpose. We computed the top 10 port statistics by flows and packets in January and March for all networks. The flows statistic is more stable and more suitable for analysing the behaviour of the network. We selected ten well-known ports that were most often observed in the statistic. Table 4.19 shows the percentage of flows that belong to these ports. We can observe that the networks have very different usage of the ports which makes port usage an important characteristic of the network. Note that we did not study day-night variance of port usage, but we expect that its fluctuation may be used to refine the results.

4.4.4 Conclusions

We have presented an analysis of network traffic measured at five different campus networks at Masaryk University. Our goal was to show that the properties of the networks can be extracted and quantified and that we can use the results to differentiate and describe the networks. We have made several observations during our analysis which affect the derivation of network characteristics.

- Flow-based statistics are more stable than byte or packet-based statistics. Therefore it is more practical to use flow statistics to track long-term changes in network behaviour.

Port / Network		UCB	ICS	MU	FI	VSS
DNS	**53**	9.7 %	30.2 %	21.5 %	12.2 %	42.7 %
HTTP(S)	**80**	9.2 %	7.4 %	20.2 %	16.8 %	5.9 %
	443	6.5 %	8.3 %	14.7 %	20.1 %	4.0 %
Mail	**25**	–	–	1.0 %	0.6 %	–
	993	–	1.7 %	–	0.6 %	–
Samba	**445**	1.0 %	–	–	–	0.7 %
SSH	**22**	–	–	1.4 %	0.3 %	–
NTP	**123**	–	0.9 %	7.1 %	43.9 %	–
SNMP	**161**	52.8 %	11.9 %	–	–	23.5 %
Telnet	**23**	1.0 %	1.3 %	1.6 %	0.4 %	–

Table 4.19: The Most Common Ports by Flows in January.

- Day-night and weekday patterns must be taken into consideration when computing network characteristics.
- Long enough samples of the traffic must be available.
- Using ratios between day and night helps to compensate for the day-night pattern and the size of the network. However, there is still a slight correlation between network size and the day-night ratios.
- To describe a network used to perform an experiment, the description of behaviour using network characteristics should be complemented with absolute volumetric information.

Our measurements have confirmed that it is possible to differentiate between networks by utilising the observed characteristics. Moreover, the campus networks showed quite stable characteristics over time even though the measurements were taken during the winter exam and spring teaching periods. Furthermore, we tested the correlation of the described characteristics and found that they are only very weakly correlated. This shows that each of the characteristics is valuable and carries information about the network. However, the presented characteristics are not by any means complete. We analysed a subset of possible characteristics derived from flow records, which can be easily measured on a network. Thus we believe that more work is required to identify other useful characteristics and utilise them to describe the behaviour of the networks.

4.5 Summary

In the first section, we presented an analysis of HTTP traffic in a large-scale environment which uses application flow monitoring with HTTP protocol processing. In contrast to previously published analyses, we were the first to classify patterns of HTTP traffic which are relevant to network security. We described three classes of HTTP traffic which contain brute-force password attacks, connections to proxies, HTTP scanners, and web crawlers. Using the classification, we were able to detect up to 16 previously undetectable brute-force password attacks and 19 HTTP scans per day in our campus network. The activity of proxy servers and web crawlers was also observed. Symptoms of these attacks may be detected by other methods based on basic flow monitoring, but detection using the analysis of HTTP requests is more straightforward. We, thus, confirm the added value of application flow monitoring in comparison to the traditional method.

The exhaustion of IPv4 address space increases pressure on network operators and content providers to continue the transition to IPv6. The IPv6 transition mechanisms such as Teredo and 6to4 allow IPv4 hosts to connect to IPv6 hosts. On the other hand, they increase network complexity and render many methods ineffective to observe IP traffic. In Section 4.2, we extended our flow-based measurement system to involve transition mechanisms information to provide full IPv6 visibility. Our traffic analysis focuses on IPv6 tunnelled traffic and uses data collected over one week in the Czech national research and education network. The results expose various traffic characteristics of native and tunnelled IPv6 traffic, among others the TTL and HOP limit distribution, geolocation aspect of the traffic, and list of Teredo servers used in the network. Furthermore, we show how the traffic of IPv6 transition mechanisms has evolved between 2010 and 2013.

The importance of IP address geolocation has increased significantly in recent years, due to its applications in business advertisements and security analysis, among others. Current approaches perform geolocation mostly on-demand and in a small-scale fashion. As soon as geolocation needs to be performed in real-time and in high-speed and large-scale networks, these approaches are not scalable anymore. To solve this problem, we propose two approaches to large-scale geolocation in Section 4.3. Firstly, we present an exporter-based approach, which adds geolocation data to flow records in a way that is transparent to any flow collector. Secondly, we present a flow collector-based approach, which adds native geolocation to NetFlow data from any flow exporter. After presenting prototypes for both approaches, we demonstrate the applicability of large-scale geolocation by means of use cases. Our prototypes have shown to be scalable enough for deployment on the 10 Gbps Internet connection of the Masaryk University.

Performing research on live network traffic requires the traffic to be well documented and described. The results of such research are heavily dependent on the particular network. Section 4.4 presents a study of network characteristics, which can be used to describe the behaviour of a network. We propose a number of characteristics that can be collected from the networks and evaluate them on five different networks of Masaryk University. The proposed characteristics cover IP, transport, and application layers of the network traffic. Moreover, they reflect strong day-night and weekday patterns that are present in most of the networks. Variation in the characteristics of the networks indicates that they can be used for the description and differentiation of the networks. Furthermore, a weak correlation between the chosen characteristics implies their independence and contribution to the network description.

Chapter 5

Flow Monitoring Performance

Monitoring of high-speed networks is becoming a resource-intensive task. There are dedicated flow monitoring probes built with commodity hardware support up to 10 G links, but multiple 10 G or even 100 G optical networks are being used for transport networks and data centre connectivity. Moreover, application flow monitoring is even more demanding than the basic flow monitoring. Therefore, the performance of flow monitoring is an important topic.

The goal of this chapter is to describe how the performance of flow monitoring can be measured and what steps can be taken to improve the performance. We describe the state-of-the-art in packet capture, which is an essential part of every flow monitoring system. We show which parts of the packet capture directly influence the design of the flow monitoring process and discuss how the flow monitoring can take advantage of the different features provided by packet capture frameworks. Description of the most impactful optimisation techniques both with and without the use of specialised FPGA-based network interface cards is provided. Moreover, several novel techniques for improving flow monitoring performance are proposed as well.

Running and maintaining separate probes to monitor multiple links is uneconomical and time-consuming. Therefore, we explore the possibility to facilitate network interface cards (NICs) with multiple 10 G interfaces to build probes which can replace many existing boxes, leading to reduced management and operational costs. The monitoring performance is critical for such a high-density solution. We use two custom-built, FPGA-based NICs, each with eight 10 G interfaces to test current CPU limits and to propose improvements for the near future commodity NICs.

The paper included in this chapter is [A13]. The paper related to this chapter is [A14].

The organisation of this chapter is as follows:

- *Section 5.1 provides background to the measurement of flow monitoring performance and highlights its pitfalls.*
- *Section 5.2 describes the state-of-the-art of high-speed packet capture, which is an important part of every flow monitoring system.*
- *Section 5.3 shows how can the flow monitoring be accelerated with the use of specialised FPGA-based network interface cards and by optimising the flow exporter software.*
- *Section 5.4 presents a high-density flow monitoring system capable of monitoring of sixteen 10 Gb links in a single box and evaluates its performance.*
- *Section 5.5 summarizes the chapter.*

5.1 Measuring Flow Monitoring Performance

The primary goal of measuring flow monitoring performance is to determine whether the system can be deployed to monitor a particular network or a link. For example, if the system is capable of monitoring 20 Gb/s of traffic, it can be deployed in both directions of a 10 Gb/s link. However, the claim that the system can handle 20 Gb/s may mean a lot of different things and it usually differs from vendor to vendor. The throughput of a flow monitoring process depends on a number of factors, such as the configuration of the flow creation process and the type of network traffic that is being monitored. 10 Gb/s may mean 14.88 Mpps (millions of packets per second) for minimum Ethernet frames (20 B + 64 B) or just 813 kpps for maximum Ethernet frames on 1500 MTU (each frame is 1538 B). The processing of different packet headers can take different time as well, for example, IPv4 vs IPv6 or UDP vs TCP. The number of flows has a large impact on the flow creation process as a flow record must be kept for each of the flows. Therefore, a detailed performance analysis of the flow monitoring system should be performed before its real-world deployment.

The throughput of the flow monitoring system is usually measured either in packets per second or bits per second. However, since the packet size should be reported as well so that these values provide meaningful information, they are mostly interchangeable. Packets per second metric is preferred in rest of this chapter.

The goal of this section is two-fold. Firstly, it describes the parameters that need to be specified when reporting on a flow monitoring system performance. There are three different categories of the parameters: hardware configuration, exporter configuration, and traffic composition. Secondly, measurement of the impact of optimisations is discussed. Although the impact can be measured simply by an increase in the overall throughput, there are a few pitfalls that need to be recognised and avoided for the results to be interpreted correctly.

5.1.1 Measuring Overall Throughput

The overall throughput of the flow monitoring process depends upon a number of factors. Moreover, there are two methods of measuring the throughput even if all of these factors are set. The first method is to generate the testing traffic at such a rate that is above the capability of the monitoring process to handle. The throughput is then determined as the number of packets that were actually processed by the flow exporter. This method gives an upper bound on the capabilities of the monitoring process, and the measurement is quite easy to carry out. However, deploying the flow monitoring system based on such an estimate can easily lead to dropped packets in practice since there is quite a large packet drop associated with the measurement as well. The second method measures throughput without packet loss or with a defined packet loss rate. Given a predetermined packet loss rate, the traffic is generated so that the flow monitoring system performs with lower that given packet loss rate. The throughput is then increased until the packet loss matches the predetermined value and the throughput at that point is considered to be the result of the measurement. Both performance measurement methods are important. The first one gives an upper bound of the capabilities of the flow monitoring process and the second one shows how much traffic can be safely processed with acceptable packet loss.

When evaluating the performance of a flow monitoring process, the hardware on which it runs must be taken into consideration. The performance depends on the frequency of the processor, available instruction sets, and size of caches. When the exporter software is able to utilise multiple threads, the number of cores of the processor and the number of processors matter as well. Moreover, when there is more than one processor, the correct use of Non-Uniform Memory Access (NUMA) architecture is important. The buffers of processes and threads running on a one CPU should be in a memory connected to the same CPU. Otherwise, the latency for accessing this memory is increased as the data must be accessed by a memory controller of another

process through a bus, such as the QuickPath Interconnect [183] bus or its newer version Ultra Path Interconnect (in case of Intel processors).

The choice of packet capture solution often limits the upper bound on the throughput of the flow monitoring process. Although packet capture is an important part of flow monitoring process, it is used for other tasks as well and is of interest to various researchers [184, 185]. High-speed packet capture is described in more detail in Section 5.2.

The configuration of the flow exporter has a significant impact on the throughput as well. The active and inactive timeouts affect the number of created flows and flow records considerably [15], which in turn affects the utilisation of the flow cache and therefore the total throughput. The flow cache size should be tuned to the expected number of flows so that the data locality is preserved as much as possible. Moreover, the size of flow records depends on the number of extracted properties, and it affects the size of the cache as well. The more properties extracted, the larger the flow record. Naturally, the number of extracted properties has a direct impact on the throughput since each property must be extracted and stored in the flow record, which requires computing power and more memory accesses. We have demonstrated the impact of the number of extracted properties on the performance in Section 3.5.

One of the most important aspects of throughput measurement is the used dataset. Choosing shortest packets and a single flow allows to benchmark the packet parser, shortest packets and a large number of flows put great stress on the flow cache. Using more complex packets with multiple protocol layers tests the packet parser capabilities and performance as well. The relative sizes of flows in the dataset matter as well. When there are a few dominant flows, the corresponding flow records are kept cached, and the overall performance increases. However, if the packets of the flows are equally interlaced, the caching of the flow records is limited. Since the choice of the dataset has such an extensive impact on the flow monitoring process and its throughput, it is imperative to choose a correct dataset that supports the intentions of the given benchmark. Guidance for choosing a dataset can be found in the work of the IETF Internet IP Performance Measurement working group [186] and IETF Benchmarking Methodology working group [187], especially in [188, 189].

We have argued that there are many factors that influence the throughput of a flow monitoring system. Since it is clearly not possible to test all combinations of the input parameters, it is necessary to choose a set of appropriate settings that cover either the most extreme scenarios or approximate some important real-world scenario. The work of Nassopulos et al. [185] shows that performing a measurement using a large number of configurations allows deriving results that can help with the optimisation effort. They show that the implementation of the flow cache as well as the composition of the traffic has a large impact on the flow monitoring process. Regardless of the chosen parameters, a proper description of the used configuration and dataset must always accompany all presented results. Therefore, we recommend using a synthetic dataset that can be described in a few simple rules.

5.1.2 Measuring an Impact of Optimizations

Throughput measurement is an integral part of any performance optimisation effort. The goal of the measurement, in this case, is two-fold. Firstly, bottlenecks need to be identified so that the optimisation can be effective. For example, if a flow cache lookup takes 70 % of the total processing time for a single packet, optimising packet parser is not very effective. Secondly, the measurement is repeated to examine the effect of a given optimisation. Moreover, it is necessary to verify that optimising for a single use case does not have a negative impact on other use cases. Therefore, thorough performance measurement is often repeated during any optimisation process.

Identifying bottlenecks can be a complicated process. It is often beneficial to measure execution time spent on processing a single packet throughout the flow monitoring process. The authors of [190] show that it is possible to roughly estimate the time required for packet reception and parsing based on several parameters. Gallenmüller et al. apply a similar methodology to compare the performance of several high-performance packet processing frameworks in [191]. Using a similar process can help to identify bottlenecks in the flow processing as well.

The processing speed of a computer is limited by the speed of the processor and the time required to fetch data from memory. If a computation requires only a small amount of memory that fits in CPU registers or L1 cache, the processor can be fully utilised. However, when a lot of data needs to be processed, there are many uncached accessed to memory with high latency. This causes the CPU utilisation to be low. Moreover, the memory bus utilisation is higher since the data needs to be transferred from RAM to CPU cache more often. Therefore, it is very useful to measure the number of instructions per processor cycle (IPC). The higher the IPC, the better the CPU is utilised. When optimising such a process, the computation should be simplified. However, when the IPC is low, the CPU waits for data from memory. In such a case, optimising data locality and memory accesses is the best way to improve the performance.

Since multi-core CPUs are a standard nowadays, an excellent way to improve performance is to process the data using multiple threads. However, it is important to keep in mind that the bandwidth of CPU memory controller is limited and that passing data between the threads consumes the bandwidth as well. Finding a right balance is often a difficult task which requires experimentation and thorough measurement of the system load.

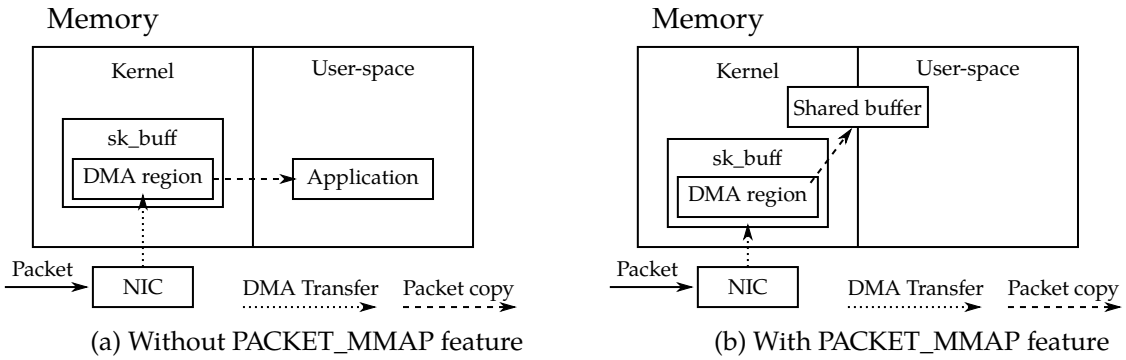
A good way to understand the relationship between CPU processing power and memory bandwidth is to use Roofline Performance Model [192]. Its core parameter is Arithmetic Intensity which is the ratio of total floating-point operations to total data movement. The visualisation of the model shows both the implementation performance limitation and limitations of the used architecture. However, correctly estimating the arithmetic intensity of the flow monitoring process is a challenging task.

Optimizations proposed in Section 5.3 either reduce the amount of processed data or reduce the number of computation steps necessary to process each packet. We also discuss multithreaded flow processing and point out the pitfalls and caveats that can result in incorrect results.

5.2 High-Speed Packet Capture

Packet capture is the first step of the flow monitoring process and has a significant impact on the overall performance. Moreover, as it is an integral part of other network applications, such as switching and routing [193], the high-speed packet capture using commodity hardware running Linux operating system has been an active research area in recent years. Since the performance of packet capture directly impacts the performance of the whole flow monitoring process, the aim of this section is to present basic principles and significant technologies in this area. Furthermore, some techniques of the packet capture, such as Receive Side Scaling (RSS), have a direct impact on the functionality of flow monitoring as well.

The goal of high-speed packet capture is to deliver packets from Network Interface Card (NIC) to the user-space application as fast as possible while providing the desired functionality, such as packet fanout to multiple threads and applications. The main difference between high-speed packet capture and standard packet processing in an operating system is that the high-speed packet capture is not connection oriented. It does not provide socket API for applications to easily communicate through. Instead, it allows applications to receive and send raw packets efficiently. However, to understand the packet capture process and its limitations, we shortly



(a) Without PACKET_MMAP feature (b) With PACKET_MMAP feature

Figure 5.1: Packet Transition Through Memory in Linux NAPI.

describe how Linux network stack receives packets, as it is the common starting point of all packet capture frameworks.

5.2.1 Linux Network Stack

The basic function of every NIC is to receive a packet from the network and pass it to the RAM where it is processed further. Direct Memory Access (DMA) is used for the data transfer. The main advantage is that it does not require the cooperation of the CPU and does not consume any CPU cycles. The NIC can access only a limited memory region for which the DMA is permitted. In the Linux network stack, this region is part of the kernel memory and is not directly accessible to the user-space. The way to notify the kernel of arrival used to be through Interrupt ReQuest (IRQ). However, as the network speed increased to 1 Gbps speed, full utilisation of the network used to cause an interrupt storm – way too many interrupts for the system to handle efficiently. The original network stack was replaced by New API (usually called NAPI), which uses polling for high-speed packet reception and falls back to interrupts for lower packet rates. This combination of both interrupt and polling approaches allows to conserve CPU cycles for low traffic and avoids interrupt storms for high packet rates. The NAPI was introduced in Linux kernel versions 2.4.20 and 2.6.

After the packet is present in RAM and the kernel is notified either by interrupt or polling, the packet is copied to internal kernel structure called *sk_buff*. Modern NIC drivers support copying of packets directly to a *sk_buff* structure, which eliminates the copying and increases performance. The *sk_buff* structure is then pushed to kernel network stack, where it is processed and made available to user applications. However, every call to a system *recv** function also needs to provide a pointer to an user-space buffer to which the data are copied. Therefore, the packet is copied at least once by the kernel, as demonstrated in Figure 5.1a. Moreover, before the *recv** system call can provide data to user-space, each packet needs to be processed by the Linux network stack. The processing often includes the whole TCP/IP stack and is quite complex and resource consuming.

The Linux network stack cannot be used for packet capture because the stack takes care of all layers of the packets and delivers only the application content to the application. However, there is an option to use raw packets where the application receives and sends complete packets with all headers, including the Ethernet. The reception of raw packets uses the *sk_buff* structure and two copy operations are needed for each packet as well. Despite that, the absence of network stack processing provides a large performance benefit. Moreover, it is possible to utilise the RSS feature provided by some NICs and receive packets in different receive queues, which helps to increase performance as well. However, the standard API for working with packets requires performing a system call for reception of each packet (two calls when a timestamp is needed as well), which causes frequent switches between user- and kernel-space.

To provide an efficient way to perform packet capture using standard Linux kernel and drivers, *PACKET_MMAP* [194] feature was added to the kernel. It utilises a shared buffer between the kernel and the user-space, as shown in Figure 5.1b. Therefore, the number of packets that can be read in a single batch is not limited, which significantly lowers the number of necessary system calls and thus improves performance. Another advantage of the PACKET_MMAP feature is that the processing of packets can be performed by multiple threads. The distribution of packets can be performed using a hash, round robin, random selection, and several other policies.

Although the Linux NAPI has evolved to support fast packet handling, there are several important performance limitations left:

- The packets are copied from DMA memory region to the user-space accessible buffers.
- The packets are always processed by kernel network stack.
- There is little support for NUMA architecture awareness.

There are several frameworks that aim to remove these limitations to achieve the highest possible throughput for packet I/O operations.

5.2.2 High-Speed Packet I/O Frameworks

Although there were several works analysing the performance packet capture applications, such as [195], one of the first alternatives to using the Linux network stack for packet capture was *PF_RING*. The PF_RING socket was created by Luca Deri and its first description appeared in [196]. The PACKET_MMAP was very new at that time and some of the introduced features, such as memory mapping overlapped. However, the PF_RING had several advantages. Firstly, the packets were not queued into kernel network structures at all. It meant that they were not available to standard applications using socket API but only to applications that used the PF_RING sockets. Secondly, packet sampling was implemented at a lower level, which meant that sampled packets were not even passed to upper layers. Lastly, multiple applications were able to open several PF_RING sockets simultaneously without interference. However, the main performance limitations remained the same as with Linux NAPI.

The problem with unnecessary copying between DMA-able memory and the shared buffer was already solved when a specialised NICs with custom drivers were used. One of the most used cards for this purpose was the DAG NIC [197] which was eventually commercialised by the Endace company [198]. The DAG NIC used FPGA chip that could be programmed to provide specific functionality, such as high-precision packet timestamping. Moreover, it meant that it was possible for the driver to instruct the card to upload packets in a correct format directly to the buffer that was shared to the user-space. Degioanni and Varenni used this approach and proposed a design for distributing packets to multiple threads to better utilise CPU parallelism [199]. However, the card did not support RSS and thus the single buffer to which the NIC uploaded the packets remained to be a bottleneck.

To eliminate unnecessary packet copying even without the use of a specialised NIC, Deri developed a new framework called nCap in 2005. The most important innovation was that the standard kernel packet processing was completely bypassed. A specialised driver was developed for compatible NICs (Intel 1 GE and 10 GE adapters were supported) that allowed the NIC to manage the packet reception and transmission buffers directly. These buffers were then mapped directly to user-space and a user-space library called nCap is used to manage these buffers. This *zero copy* approach, shown in Figure 5.2, is used in almost all other high-speed packet frameworks as well. The main disadvantage is that it requires specialised or at least modified drivers and a supported NIC. The direct access to packet buffers filled by the NIC

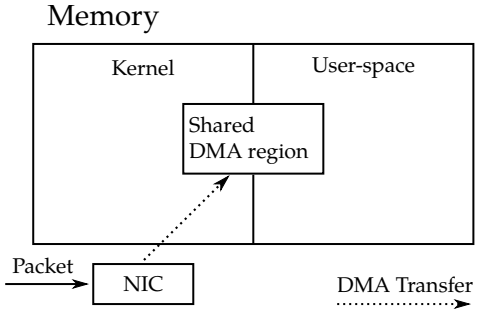


Figure 5.2: Packet Transition Through Memory Using a Zero Copy Approach.

was eventually combined with PF_RING into the PF_RING DNA [201] (Direct Memory Access) framework. The disadvantage of this approach is that only one application at a time can access the packet buffer.

When commodity NICs started to support Receive Side Scaling (RSS), the packet capture frameworks started supporting this feature to utilise multi-core CPUs better. Fusco and Deri showed how RSS could be utilised by multi-threaded polling together with the zero copy approach in 2010 [202]. The basic idea was to use one kernel polling thread for each receive (RX) queue and make the associated buffer available to one user-space capture thread. This approach, called *TNAPI*, was used together with the PF_RING framework and utilised a custom driver for zero copy packet reception. The authors showed that it is important to correctly configure the system to work on NUMA architecture by setting CPU affinity of the individual polling and packet capture threads. Therefore, this approach mitigated the limitations of Linux NAPI discussed in the previous section.

The development of PF_RING, nCap, DMA, and TNAPI frameworks resulted in the creation of the *PF_RING ZC (Zero Copy)* framework [203]. It uses proprietary drivers to deliver packets to PF_RING using the zero copy approach, supports RSS and NUMA architecture to spread packet processing across multiple CPU cores and threads, and allows the kernel networking stack to process packets when no PF_RING aware application is running. Moreover, by managing the PF_RING in the kernel, it supports packet fanout for multiple user-space applications and even virtual machines. The PF_RING ZC also allows user-space applications to receive packets in bursts (batches), which was not possible in vanilla PF_RING. This feature allows to reduce the number of system calls and significantly helps to increase performance, especially after the mitigations to Meltdown and Spectre attacks made syscalls more expensive [204].

PFQ and *netmap* packet capture frameworks were published 2012. Around the same time, Intel started public development of the *Data Plane Development Kit*. The *PFQ* packet capturing engine was introduced by Bonelli et al. in [205]. PFQ supports RSS, both vanilla and modified drivers (for zero copy operation), and utilises packet processing within NAPI context. The transition of packets from kernel-space to user-space is implemented by a wait-free queue. When packets need to be copied, it is done in batches, which increases performance. The authors showed that the performance of PFQ is better than that of vanilla PF_RING in their setup. PFQ has evolved significantly in the recent years. The authors added many features [206, 207, 208], such as support for in-kernel programmable early processing and user-space parallel processing while preserving normal host system network stack operation.

The *netmap* framework developed by Rizzo is different from PF_RING and PFQ in a fundamental way. The packets are not stored in continuous buffers but rather in separately allocated buffers of fixed sizes. It allows passing packets between interfaces in a zero-copy manner. Moreover, a user-space application can exchange packets with the NIC and the host networking stack independently. This design is useful for applications such as packet routing and forwarding.

Intel's *Data Plane Development Kit* [210], known simply as *DPDK*, is a set of libraries and drivers for fast packet processing. After Intel started the project, its governance came under the Linux Foundation Project, which unites many companies that contribute to the project. Similarly to netmap, it uses discretely allocated packet buffers instead of a buffer ring. However, user-space communicates directly with the card to eliminate unnecessary system calls. Therefore, when a driver with DPDK support is loaded, the NIC cannot be used together with the standard kernel networking stack. By utilising per packet buffers, DPDK allows processing packets out of sequence, which is useful for example for a selective packet capture. Matching packets can be retained while others are quickly discarded. This is not possible with ring buffer approach without packet copying. DPDK does not support interrupts; therefore packets are always polled from the NIC. This is effective to high loads; however, it excessively utilises the CPU for lower packet rates. DPDK is widely supported and specialised drivers exist for many NICs from different vendors [211].

There are other packet capture and manipulation frameworks that were not discussed in this section, such as OpenOnload [212] from Solarflare and PacketShader I/O [213] using GPU acceleration to build fast router. Moreover, manufacturers of (FPGA-based) high-speed NICs such as Endace, Napatech, Myricom, Mellanox Technologies, or Netcope Technologies provide their own drivers and libraries for fast packet processing together with their products. These drivers and libraries utilise very similar, if not the same, approaches like the ones described in this section.

There are several works comparing the performance of different packet capture frameworks. They can be used as a reference when choosing the correct framework for a particular use case. Braun et al. evaluate the performance of native FreeBSD and Linux packet capture together with PF_RING on Linux [214]. They identify bottlenecks and provide guidelines for debugging loss of performance. The authors of [184] not only compare the performance of PacketShader, PFQ, and PF_RING, but they also provide a detailed description of these frameworks and Linux NAPI as well. Wang et al. compare PF_RING, PF_RING ZC, and DPDK frameworks in [215]. The focus of this comparison is on the correct utilisation of the NUMA architecture. The authors conclude that the difference in thread assignment between the compared frameworks has an impact on the overall performance. However, they fail to realise that processing on one of the NUMA nodes must be naturally faster since the NIC is directly connected only to this one node. The authors of [216] not only compare features and performance of PACKET_MMAP, PacketShader I/O, netmap, PF_RING ZC, and DPDK, they also analyse the use of these frameworks with modular router Click. They extensively discuss the impact of important features such as I/O batching, ring size, execution model, zero-copy, and multi-queuing on the final performance of the Click router. Finally, Gallenmüller et al. propose a model to estimate and assess the performance of various packet processing frameworks in [191]. They apply the model to DPDK, netmap, and PF_RING ZC. They assess the transmission efficiency, the influence of caches, the influence of batch sizes, and latency for each framework.

5.3 High-Speed Flow Monitoring

Packet capture is only the first part of the flow monitoring process. After the packet is received by the flow exporter software, it needs to be processed as quickly and efficiently as possible. Packet processing, flow creation, and flow export have been described in Chapter 2 and this section aims to provide an overview of techniques that can be used to accelerate these elements of the flow monitoring process.

We start with a description of techniques that rely on hardware acceleration which uses special features of the NIC. These features include receive side scaling discussed in the previous section and packet preprocessing. Software acceleration techniques are available regardless of

Hardware acceleration	Software acceleration
Receive Side Scaling	Multithreading
Packet trimming	NUMA awareness
Packet header preprocessing	*Flow state in parsers*
Flow processing offloading	Flow cache design
Application identification	Per-flow expiration timeout
	Delayed packet processing
	Bidirectional flow records

Table 5.1: Flow Acceleration Techniques (*Novel Proposals Are Highlighted*).

the used NIC and involve effective processing on multiple cores and flow creation design. The Table 5.1 provides an overview of the discussed techniques. The highlighted techniques are novel and represent our contribution to the high-speed flow monitoring field. We deliberately avoid the use of sampling since it degrades the quality of exported data, as shown by the authors of [7].

This section provides an overview of the flow monitoring acceleration techniques. However, we do not have explicit data to show the direct impact of the presented techniques on the flow monitoring performance. A case study utilising several of the presented techniques is presented in Section 5.4 instead. This study shows that it is possible to achieve more than 100 Gb/s throughput on a commodity CPU when hardware-accelerated NICs are used and several acceleration techniques are applied.

5.3.1 Hardware Acceleration in Flow Monitoring

Hardware accelerated flow monitoring is based on offloading of parts of the packet or flow processing to the specialised hardware, mainly the NIC. The extent of the offloaded work depends on the capabilities of the NIC. A field-programmable gate array (FPGA) is usually used in hardware-accelerated network interface cards. It allows to parse packet headers and perform tasks such as packet trimming, computing hashes to identify flows, or timestamp assignment directly in the NICs. It usually supports transferring data to multiple buffers in RAM so that the software can efficiently use a zero copy multi-threading model. As described in the previous section, there are several manufacturers that provide these cards. However, NICs can support hardware acceleration even without an FPGA chip. The capabilities are usually more limited and cannot be extended after the card is produced. For example, Intel produces many NICs, such as the Intel XL710 [217], without an FPGA chip, which provide basic hardware acceleration such as packet hashing or even timestamping. Use of commodity cards with advanced features like timestamping or packet filtering has been proposed by Deri et al. in [218] in 2013.

We are not the first ones to propose the use of hardware acceleration in flow monitoring. Antichi et al. proposed to use NetFPGA platform for passive network measurement in [219]. They note that without hardware support, the packet timestamps have poor accuracy. The use of FPGA allows to timestamp, filter, or trim the packets, which saves CPU processing time and allows achieving higher packet rates. However, their work is targeted only on 1 G environment.

Receive Side Scaling

Receive Side Scaling has support in almost all modern NICs. It is used by Linux NAPI to store the received packets to different queues in order to utilise multiple threads for packet processing. The NAPI ensures that packets belonging to a single application always arrive in the correct

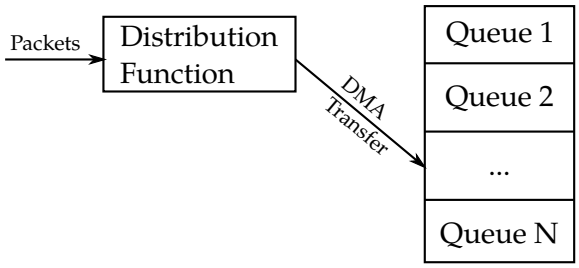


Figure 5.3: Receive Side Scaling Packet Distribution.

order even if received by different queues. However, the packet capture frameworks do not usually provide such a guarantee.

Figure 5.3 shows the principle of the RSS. Each packet is processed and a receive (RX) queue is selected. It allows multiple threads to process packets from different queues without any synchronisation, therefore the choice of the distribution function crucial. Many applications, including flow processing, require all packets of the same flow to be processed by the same thread. Therefore, the distribution function must put the packets of the same flow to the same RX queue. To achieve this, the distribution function is usually implemented as a hash function that computes a hash from certain header fields of each packet. Note that in case of flow monitoring, these fields must be a subset of flow key; otherwise, the flow could be inadvertently split into several RX queues.

IP addresses and transport layer ports are usually used to compute the hash. However, it is often forgotten that fragmented packets do not always have transport layer information present, which can result in fragmented packets being sent to different queue than the rest of the flow. Therefore, it is better, in most scenarios, to use only IP addresses to distribute the flows to different queues.

Sometimes the application needs to process both direction of a communication, e.g., biflow. In such a case, the used hash function must return the same value for both directions. We have proposed to simply order the bidirectional fields (IP address and ports) by value before applying the hash function. This solution is implemented in the HANIC design [220] of COMBO cards family. Intel NICs allow setting of different RSS hash keys to achieve different hash distribution. Although there are several symmetric RSS hash keys that create the same hash for both directions of the communication, the hash distribution is compromised [221]. Therefore, if the hash is used to identify the flow afterwards, there are many collisions. We found in our experiments that it is more effective to recompute the hash in software in these cases.

A problem that is often encountered when using RSS is that each NIC supports only a specific set of link layer protocols. When an unknown protocol is encountered, all packets that cannot be parsed are sent to a default queue. Therefore, RSS cannot be effectively used for these protocols. FPGA-based cards can be updated with new firmware; however, commodity cards with a fixed set of features should be carefully chosen when an uncommon link layer protocol is used.

The performance improvement achieved when using multiple queues is directly related to the number of queues and the number of available CPU cores. Since most queues are going to be actively polled under high load, using more queues than CPU cores available is not advisable, as it causes the threads to be switched on the CPU. However, too many queues also induce a high load on a memory controller causing the performance to be degraded. It is best to experiment with different numbers of queues to find an optimal value for the target system. We have achieved the best results with 8-16 queues, depending on the specific scenario and the number of CPU cores.

Packet Trimming

A simple way to lower the memory controller load is to trim the received packets when packet payloads are not required, e.g., for basic flow monitoring. 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. A similar technique can be used when copying packets from kernel to user-space as shown in [196]. The author uses libpcap with snaplen 128 B which speeds up packet capture.

However, much better acceleration is achieved when the packets are trimmed by the NIC and only the shortened packets are copied to the RX queues. Although the per-packet processing costs are usually not decreased, this technique significantly helps the memory controller by lowering the data throughput.

The performance improvement achieved by this method depends on the average length of processed packets. For example, the shortest packets cannot be trimmed at all; therefore packet trimming would not help in some artificial benchmarks. Moreover, application flow monitoring 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 monitoring.

Packet Header Preprocessing

Basic flow monitoring usually utilises only a small subset of packet header fields. Therefore, a 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 allows the packet processing in the flow exporter to be simplified since it only needs to read the data from a well-defined structure. Moreover, it also helps the memory controller by sending only the necessary subset of all available data.

This process requires very specialised FPGA-based cards, such as the COMBO family cards from CESNET. We have successfully used the proposed technique method to build a high-density flow monitoring system. The results are presented in Section 5.4.

Even better results could be achieved by the close collaboration of the NIC and flow exporter. If the NIC could send the data in the exact format that is used by the flow exporter, it could be used directly without reordering the elements and filling out internal structures of the flow exporter. In any case, this method is not applicable for application flow monitoring since application layer parsing is too complex and dynamic to be fully handled in the FPGA.

Flow Processing Offloading

We have shown that packet processing can be offloaded to the NIC almost entirely. However, it is possible to go even further and offload the flow creation process to the NIC as well. It was proposed by Zadnik et al. in [222] to create a flow cache directly in the NIC. However, this approach is not used in practice as it has two main disadvantages. Firstly, the memory requirements of the flow cache exceed the resources available in most NICs even today. Secondly, the flow exporters often support additional on-demand features such as processing of application layer information, or extended monitoring of TCP connection features. These features are hard or even impossible to implement in the firmware of the NICs. To phrase it differently, the NICs are not flexible enough to efficiently keep up with changing demands placed on the flow monitoring.

We have established that packet header preprocessing and packet trimming 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. Moreover, once the important features of each flow are extracted, the rest of the flow creation process usually consists of simply incrementing various counters. Therefore, it is possible to offload this part of the processing to the NIC. Such a solution is proposed in [223] 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 N^{th} 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.

Although the SDM is the most efficient solution for offloading part of the flow processing to the NIC, there are disadvantages 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 the HTTP protocol, the HTTP pipelining cannot be detected and information about further requests and responses in the same connection is lost. Therefore, SDM cannot offload flows that are expected to have important data in the payload of future packets. Another problem can occur when an offloaded flow record is expired by the flow exporter, e.g., due to a collision in the flow cache. In this case, the NIC is instructed to start sending full packets for this flow again. However, there is a delay in this process and some preprocessed packets might already have been sent to an RX queue. Without an existing flow record, partial information from the preprocessed packets cannot be used to create a new flow record. The reason is that it might be missing some of the link layer information, which is considered to be constant for each flow an thus does not need to be updated with each packet.

Application Identification

Packet classification in FPGA is not a novel idea in itself [224], however, it was not yet used together with high-speed application flow monitoring. Since only a small portion of all packets carry application protocol information necessary for flow monitoring, it is beneficial to recognise such packets in the NIC and mark them for further processing in the software. As the NIC cannot do complex application header processing, only a simple mechanism, such as pattern matching, can be utilised for packet classification. The work of the WAND Research Group [100] 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 preprocessing of such a flow can be terminated and full packets returned to the software. This approach would effectively solve the above-mentioned 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.

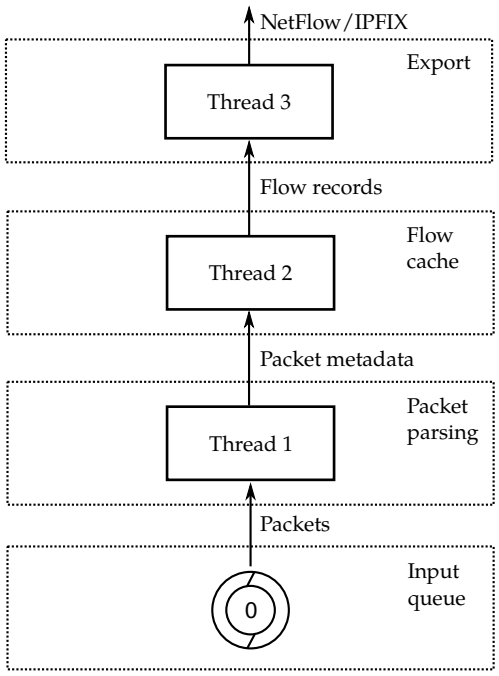


Figure 5.4: Multithreaded Flow Monitoring on a Single Input Queue.

5.3.2 Flow Exporter Software Optimization

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. Therefore, we focus on flow monitoring optimisations that can be achieved by careful software design and system configuration in this section.

Multithreading

The development of modern CPUs has a tendency to substitute raw power (frequency) for a higher number of CPU cores. To fully utilise the potential of a multi-core CPUs and achieve high flow monitoring performance, it is necessary to create a multithreaded flow monitoring software.

The three main parts of flow monitoring process can be easily executed in different threads. The first thread receives packets and parses the packet headers. The parsed data are passed to the flow creation thread that manages the flow cache. When a flow is exported from the flow cache, it is passed to the third thread that manages flow export. Figure 5.4 illustrates this execution model, which uses 3 threads in total.

For NICs with RSS support, the flow processing can utilise more than a single queue. The easiest approach is to have different threads process packets from different buffers. The resulting flow records are aggregated either in a single flow cache or multiple flow caches, one per each thread, as shown in Figure 5.5. 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 as previously described in the passage about Receive Side Scaling. Using this approach, $N + 2$ threads in the first case or $2N + 1$ threads in

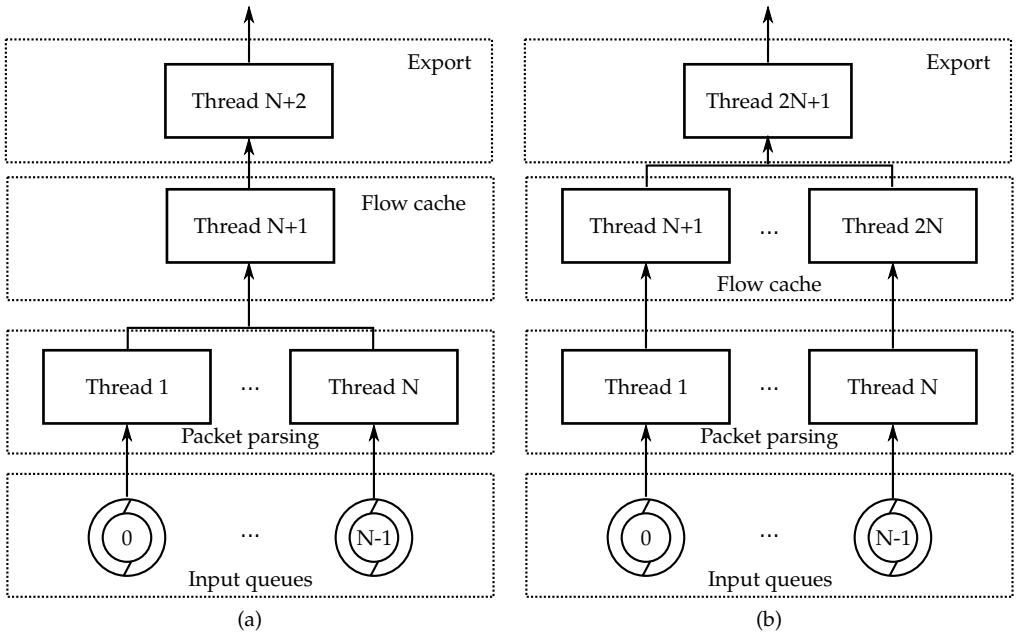


Figure 5.5: Multithreaded Flow Measurement with a RSS Support: a) Single Flow Cache; b) Multiple Flow Caches.

the second case can be utilised, where N is the number of buffers. The last thread is used for flow export.

Using $2N + 1$ threads for flow processing results in 17 threads for 8 RX queues. On a system with a lower number of cores, it is desirable to use fewer threads so that each active thread has a dedicated core for itself. This helps to avoid the necessity to execute different processes and threads on the dedicated cores, which increases performance. Moreover, having the same data to be processed by multiple threads increases the impact on the memory controller, which can become overloaded for a large number of threads. Therefore, lowering the number of cores per RX queue can, in fact, help the performance. By combining packet processing and flow cache to a single thread, the total number of threads can be lowered to just $N + 1$ while retaining multiple flow caches. Whether to combine the threads or not depends on the throughput of the memory controller, CPU frequency, number of cores, and the actual amount of work performed by each thread. The measurement framework proposed by the authors of [191] can be used to determine the characteristics of the system.

Another reason to keep packet parsing and flow cache in a single thread is that the packet parsing and processing of application protocols may require a state of the connection to be kept. In this case, the flow cache record must be made available to the packet processing thread. If the threads were to be kept separate, it would result in a use of synchronisation 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. However, it is much more simple to use the flow cache directly when the packet parsing and flow cache are managed in a single thread.

NUMA Awareness

To build a high-speed flow monitoring system, servers with multiple CPUs are often used. However, special care must be taken to deploy such a configuration properly. Modern systems use Non-Uniform Memory Access (NUMA) architecture where each CPU has direct access to single *local* memory bus and a high-speed interconnect (such as QPI) must be used for communication

between different nodes (CPU reading from a *remote*, without a direct access). Moreover, each NIC is directly connected to only one CPU. Therefore, allocating DMA buffers for the packet reception in a remote memory is ineffective as it needs to use the high-speed interconnect between the CPUs.

The generic rule is to avoid using remote memory as much as possible. Therefore, the buffers for the RX queue should be allocated on the CPU connected to the NIC. However, when multiple CPUs are used for processing packets, it makes sense to distribute the buffers to those CPUs. The threads processing packets should always run on a local core, which can be achieved by setting the affinity of the thread to the CPU of the specific core. When starting new core, it is important to set its affinity first and only then allocate memory, because the memory is allocated on the local core by default. If the thread is migrated to a different CPU, the local memory becomes remote.

If biflow creation is not required, it is possible to utilise two NICs to process different directions of the traffic. Each can be connected to different CPU, which eliminates the need to share data between CPUs almost entirely.

Flow State in Parsers

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. Furthermore, 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.

When the flow state is known to the parser, an optimisation similar to the SDM can be deployed. When an application parser detects that it has already obtained all important information from a given flow, it can instruct the exporter to skip processing of the application layer for this flow. This way the different application parsers can opt-out from processing individual flows.

This optimisation can be applied especially to the encrypted traffic using SSL/TLS protocol. It is quite easy to detect TLS encryption from the specific format of its handshakes. Without the means to decipher the traffic, other application parsers cannot process encrypted packets. Therefore, all other applications parsers can be skipped when the SSL/TLS protocol is detected. Since HTTPS adoption is steadily increasing [225], the amount of SSL/TLS connections rise as well; therefore this optimisation is likely to gain impact in the future.

Flow Cache Design

One of the most performance critical parts of any flow monitoring software is the flow cache. The cache needs to be designed to hold hundreds of thousands of flow records, do fast inserts, lookups, 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 [226].

The flow cache design has been studied in the literature [60, 185]. Although authors propose to use various data structures such as linked lists, trees, or multidimensional hashing table, our experience using a hash table indicates that the simplest structure provides the best performance. 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 the simple hash table. To avoid collisions, the flow record can be placed in any of the N positions after the computed one. Therefore, the record is always found in constant time and no overflow structures are necessary. However, the flow cache utilization never effectively reaches 100 percent as that would cause too many collisions.

It is best to estimate the maximum number of flow records that need to be kept at a time and choose the cache size so that there are no or only few collisions for this number of records. Each collision necessitates a forceful export of a flow record, which leads to performance degradation.

Implementation of active timeouts is straightforward. Each time a packet is being added to a flow record, the start time of the flow record is compared to the packet timestamp. When the difference is greater than active timeout, the flow record is exported and a new flow record is created in its stead. However, inactive timeouts are more complex to implement effectively. Different algorithms using time-ordered linked lists and last least recently used structures were proposed in the literature. However, management of such structures is costly. A straightforward approach would be to start an auxiliary thread that would search the cache for any flow records that need to be expired. However, this would cause heavy locking on the flow cache, which is undesirable as well. The the most efficient way to implement inactive timeouts on a hash table flow cache is to have the thread managing the flow cache check several records each time a packet is received. This approach avoids locking the flow cache and performs well. However, the export of inactive flow records may be slightly delayed. Moreover, the search for expired flows must be performed even when no new packets are received. Carefully balancing the flow cache management tasks is a complex problem which offers a considerable potential for further research.

The length of flow records that are kept in the cache has an impact not only on the size of the flow cache but the performance as well. Having too long flow records causes more cache misses during while working with the cache. For example, when an URL is extracted from an HTTP application header, it can be thousands of bytes long. Reserving a place for this kind of information in the flow record is impractical and hurts performance. Our measurements show that it is faster to dynamically allocate memory for application data of certain size and store only a pointer in the flow record itself. The less common the application is, the more beneficial is to allocate the space for its data dynamically since only a small amount of these allocations is performed in comparison to every flow record being significantly larger.

Per-Flow Expiration Timeout

Flow cache inactive timeout expiration has a large impact on the performance, as discussed in [62, 56]. However, the value for the timeout is usually the same for all flow. The authors of [62] show that different classes of traffic could use a different inactive timeout. To implement this in application flow monitoring, we propose that the timeouts should be configurable per flow. When a flow is successfully processed by an application parser, a timeout appropriate for this application should be assigned. This approach should decrease the memory requirements and increase the overall performance of the flow monitoring.

Flows that comprise only of a single packet represent a special traffic class. If the flow exporter is able to detect and decide that the first packet is also the last one, the flow can be directly exported and does not need to enter the flow cache at all. This is especially useful to cope with DDoS attacks, as described by Sadre et al. in [226]. Moreover, it could be used for monitoring of DNS traffic as well since most of the request and responses consist of only a single packet.

Delayed Packet Parsing

Processing application layer is often a complex task that needs to be performed in the most performance sensitive situation, during packet parsing. However, the extracted information is often just stored in the flow record and is not used in the flow monitoring process itself. We have already discussed that in some cases it is preferable to allocate memory for application data dynamically. We can take this concept further and extract only the necessary information during packet parsing (e.g., HTTP method) and store the pointer to the copy of the rest of the

application data from the packet. The rest of the data can be processed in a less critical section of the flow monitoring process, such as in the flow export thread.

The delayed packet parsing improves performance especially in cases when the application data are too large to be kept directly in a flow record and memory allocation is performed in any case. As always, the impact of this optimisation needs to be carefully examined as it depends on the utilisation of individual threads and composition of the monitored traffic.

Bidirectional Flow Records

Biflow processing can improve performance. The reason is that the shared data (IP addresses, protocol, and ports) are stored only once, which reduces memory footprint. Moreover, since both sides are often communicating at the same time, there is an improved chance that the bidirectional flow record will be still cached by the CPU. Even when uniflow is required by the flow data processing software, the biflow can be used and then converted to uniflow during the flow export process.

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

5.4 High-Density Flow Monitoring

Current commodity hardware for high-speed network traffic flow monitoring usually consists of a PC with one or two network interface cards (NICs) with a 10 G Ethernet interface. The role of the hardware is merely to capture the data from the network. The hardware setup is supplemented by open- or closed-source software performing the subsequent steps of the flow monitoring process.

While one can deploy multiple such probes for monitoring of multiple lines, it may not be the best option because of increased power consumption, a large rack space footprint and a general complexity of the monitoring system management. Our work, therefore, focuses on exploring the scalability of this concept in a single PC from the performance point of view. There are obviously concerns about various possible bottlenecks of a single PC setup. To name a few: throughput of the NIC, system bus (i.e., PCI Express) and PC memory subsystem, the performance of a single CPU core, the overhead of potential inter-core or inter-CPU communication.

By using custom-built NICs, we are able to perform experiments with the speeds beyond what is available in the commodity hardware market. The programmability of our FPGA-based NICs allows us to examine the effect of various additional hardware features, such as a packet header preprocessing or a packet trimming. We suggest that some of these features should be included in the next generation of commodity NICs to aid the performance of future network traffic monitoring systems.

The aim of our work described in this section is to identify bottlenecks and potentials for new features of current and near-future monitoring systems, as well as to demonstrate limitations of current CPUs and generally the PC architecture in the particular use case of network flow monitoring. We test a monitoring setup that can be achieved using commodity NICs; then we present impacts of the features provided by our NIC. Last but not least, we compare two different CPUs in the same conditions to determine the influence of CPU choice on the monitoring performance.

This section is structured as follows: Subsection 5.4.1 describes our monitoring architecture and how it is different from current commodity NICs. Subsection 5.4.2 explains our experiments

and methodology, while Subsection 5.4.3 presents the obtained results. Subsection 5.4.4 draws conclusions from the results.

5.4.1 Monitoring Architecture

This subsection describes our monitoring setup from hardware to software. We briefly introduce our custom-built FPGA-based NIC architecture. We utilise these NICs to incorporate multiple 10 G interfaces in one box.

Hardware

We use the COMBO-80G card [227] to receive the network traffic. The card is equipped with two QSFP+ cages, which can be set to 4×10 G Ethernet mode each, thus creating eight 10 G interfaces in total. The packets undergo a configurable processing in the FPGA and are sent to the host RAM via the PCI-Express gen3 x8 bus. The theoretical throughput of this bus is 64 Gbps, but due to the protocol overhead, the real-world throughput is slightly higher than 50 Gbps. Figure 5.6 shows the functional scheme of the described hardware.

The FPGA firmware of COMBO-80G has several features that set it apart from conventional NICs and help to improve general throughput when receiving packets. The card automatically assigns a 64-bit timestamp to each packet at the moment of reception. The timestamp has a resolution of one nanosecond, which is better than what can be achieved by assigning it later by the software application. Also, the processor load associated with the timestamp generation is removed.

Another feature is a configurable packet trimming. The card can be set to trim the packets to a predefined length, thus saving the PCI Express and memory subsystem bandwidth, most notably for long packets. This feature is clearly intended for the purpose of flow monitoring since the relevant information (packet header) is at the beginning of each packet.

Further extension of this feature leads to packet parsing directly by the card and transferring only the parsed packet header fields to the host RAM. In addition to bandwidth saving, the processor load is also reduced, because it no longer needs to perform the packet parsing operation. The parsed fields are sent in the so-called Unified Header (UH) format which has fixed structure and thus removes the need for complicated parsing conditions in the corresponding processor code.

To better utilise current multicore CPUs, the firmware features an option to distribute the packets into eight independent DMA channels. Target channel number of each packet is computed by hashing several fields of the packet header, such as IP addresses, protocol, port numbers. This ensures that there is a consistent mapping of network flows to the DMA channels so that the software threads always see complete flows. In common traffic with a large number of flows, the traffic is evenly distributed among the DMA channels.

Software

The DMA transfers themselves are simplified and optimised to bypass an OS kernel network stack. Large ring buffers are allocated in RAM during the OS driver initialisation, and the packets are then uploaded to these buffers by the card almost autonomously. The only communication between the driver and the card is an exchange of buffer start and end pointers (which mark an empty and free space in the buffer), and configurable interrupts generated by the card. The OS driver never touches or even copies the packets; it only maps portions of the buffers to user-space applications. This way the overhead of software processing is minimised and maximum CPU time is left for the application itself. Directly mapping the ring buffers memory to user-

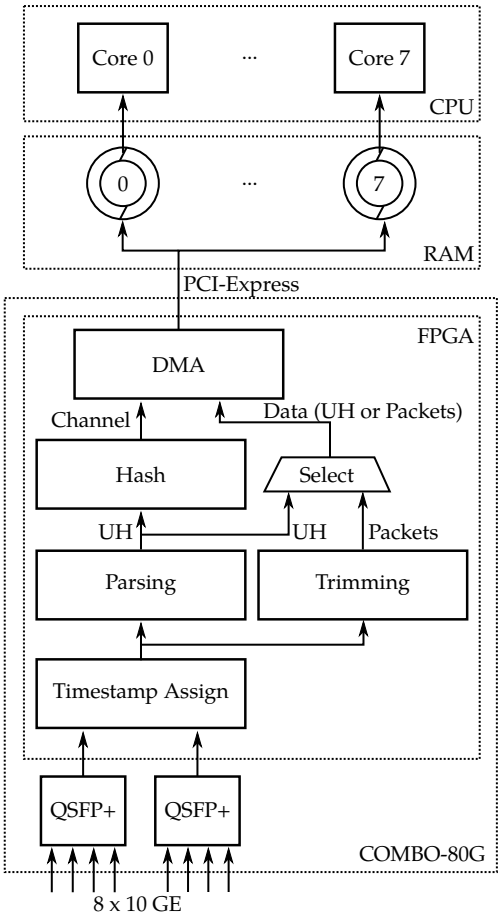


Figure 5.6: Hardware Scheme of Single Card Setup.

space also ensures that the data are copied only once, from the NIC to the RAM. This decreases the load of the memory subsystem, which helps to increase the overall performance.

Data from the ring buffers are processed by a flow exporter. We use FlowMon exporter [99] software which utilises a multithreaded design to distribute the computational load on multiple CPU cores. The flow exporter architecture is shown in Figure 5.7. Input threads read packets from ring buffers and parse L2 to L4 headers to create partial flow records. These flow records are passed to a flow cache which performs flow record aggregation. Expired flow records are exported using single unifying thread, which accepts the flows from all flow caches. A single thread is enough for this task since it is much less performance demanding.

5.4.2 Methodology

This subsection describes the setup for our experiments. We present our testbed as well as network traffic data that were used to measure the monitoring performance. Successfully processed packets per second and CPU utilisation are measured to quantify the performance.

Testbed Setup

The testbed consists of two devices. The first is the flow monitoring probe and the second is a packet generator which generates traffic that is measured by the probe.

We use Dell PowerEdge R720 server with two Intel Xeon E5-2670 v1 CPUs as the flow monitoring probe. Each CPU features eight physical cores (16 with hyperthreading), 2.6 GHz operating frequency (3.0 GHz to 3.3 GHz in Turbo mode) and 20 MB of cache. Maximum TDP of each

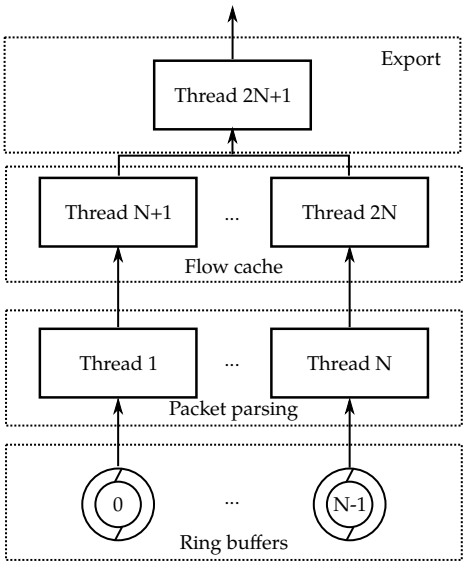


Figure 5.7: Flow Exporter Multithreaded Architecture.

CPU is 115 W. The memory controller has a throughput of 51.2 GB/s. Each CPU has four RAM modules available, running at 1600 MHz with the capacity of 8 GB per module (64 GB total). Two COMBO-80G cards are used to receive and process packets. The operating system is standard Scientific Linux release 6.5 running kernel version 2.6.32-431. The whole setup is a 2U standard rack mount PC.

Since the probe has two CPUs, we need to consider NUMA architecture specific setup. Each CPU has a directly accessible portion of RAM, which corresponds to the four physical RAM modules associated with the CPU. Accessing the memory of the other CPU is more costly since the data need to be passed between the processors using QuickPath Interconnect bus (QPI) [183]. Each PCI-Express bus is also connected to one CPU. This CPU receives the interrupts of connected devices and the devices can write directly to the associated memory using DMA transfers. Therefore, the optimal setup is to have each COMBO-80G card on the PCI Express bus connected to different CPU. The memory allocated by the drivers for the NIC should belong to the same CPU and the flow exporter should also run on this particular CPU. This way, we can almost entirely avoid QPI communications when processing the network traffic, which leads to higher performance. However, the current version of the NIC drivers does not support NUMA specific memory allocation, which causes inefficient memory access through the QPI bus. The impact of this deficiency is described in Subsection 5.4.3.

We run one instance of flow exporter on each physical CPU. This allows each exporter to use the memory of this CPU for the flow cache and therefore avoid accessing the flow cache data through QPI.

Data Generator Setup

The test traffic is generated by Spirent TestCenter hardware generator at the speed of 10 Gbps and is replicated to all sixteen input ports. Since the data on all 10 G ports are the same, we use interface numbers in the flow creation process to ensure that the same packets from multiple interfaces will create different flows. It also ensures that the timestamps seen by flow the exporter are monotonous.

We use an artificial traffic pattern for the flow monitoring performance measurement. This approach has several advantages. Firstly, we can test the worst case scenario using short packets to achieve the highest packet per second ratio. Secondly, it is easy to repeat the tests in another

Packet size	Flow count	Packets/s	Bits/s
64 B	128	14,880,545	7,618,839,040
64 B	16,384	14,880,545	7,618,839,040
64 B	131,072	14,880,545	7,618,839,040
128 B	128	8,445,715	8,648,411,900
128 B	16,384	8,445,715	8,648,411,900
128 B	131,072	8,445,715	8,648,411,900
256 B	128	4,528,861	9,275,108,100
256 B	16,384	4,528,861	9,275,108,100
256 B	131,072	4,528,861	9,275,108,100
512 B	128	2,349,560	9,623,786,500
512 B	16,384	2,349,560	9,623,786,500
512 B	131,072	2,349,560	9,623,786,500

Table 5.2: Combinations of Packet Lengths and Flow Counts Used in the Tests.

laboratory. Moreover, it is infeasible to simulate a real traffic using packet generators, especially at 10 Gbps speed. To compensate for the different number of concurrent flows and real traffic distribution of packets in flows, we repeat the tests with several different flow counts. All generated data are simple UDP packets, and flows are created by permuting parts of IP addresses. The results provided in Subsection 5.4.3 are averaged from several measurements taken for each scenario.

Table 5.2 shows combinations of packet sizes and flow counts used in our measurements. Corresponding packet and bit rates are also shown. Various bit speed is caused by the Ethernet protocol overhead, which is lower for longer packets. All values are given for a single 10 G packet generator. Since the traffic is repeated to every input interface, the actual flow count and traffic rates sent to our monitoring device are 16 times higher.

The flow count has a high impact on the flow cache performance. With the increasing number of active flow records, the memory is accessed more randomly and the CPU experiences more cache misses, which results in a higher latency of memory access and therefore in lower performance. It is difficult to estimate the number of flows to simulate from the real network since the real flow records are not updated periodically and the number of active flows in the flow cache at any given moment depends heavily on active and inactive flow cache timeouts and other software settings. Therefore, we test several different options to estimate the influence of flow count on the overall performance. We believe that the highest number of flow count used in our tests is more performance-demanding than real traffic in most networks.

5.4.3 Results

The results of our experiments are presented in this subsection. We show the measurement performance in a setup achievable by commodity NICs, and then we present the improvements achievable by our NICs. Moreover, we show a difference in performance between two different CPUs. We use packets per second as a measure of system performance. Bytes or bits per second can be gained easily by multiplying the number of packets per second by their respective sizes.

Basic Performance

First, we have measured the basic performance of the flow exporter on full packets. Figure 5.8 shows packet rates separately for each of the cards. We group the rates for the same packet lengths together. Each group has three pairs of columns. Each colour represents a different number of flows per interface. Shades of the colour differentiate the NICs. The *Maximum Ethernet* column is the highest achievable aggregated throughput of all 16 Ethernet input interfaces. The *Maximum PCIe* column has been measured by counting the packets received by the simplest possible software application (a packet counter). This way, we get an upper limit on the number of packets, that the flow exporter can receive via the system bus.

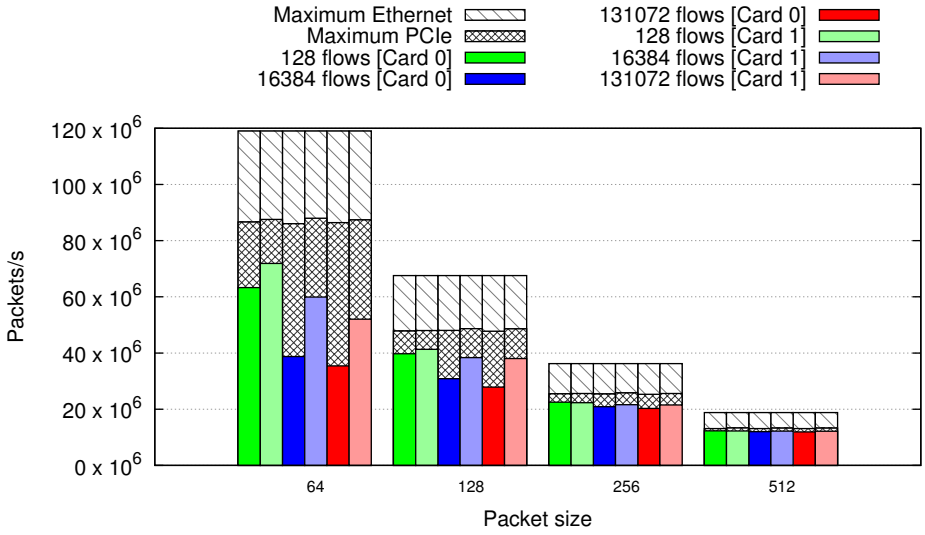


Figure 5.8: Full Packet Processing Performance in Packets/s.

We plot the CPU utilization in Figure 5.9. The interpretation of the graph is almost the same as for Figure 5.8 with the exception that values for both cards are summed up and the utilisation is shown for different flow exporter threads instead. The darker colour represents the input thread and the lighter colour marks the values for the flow cache thread.

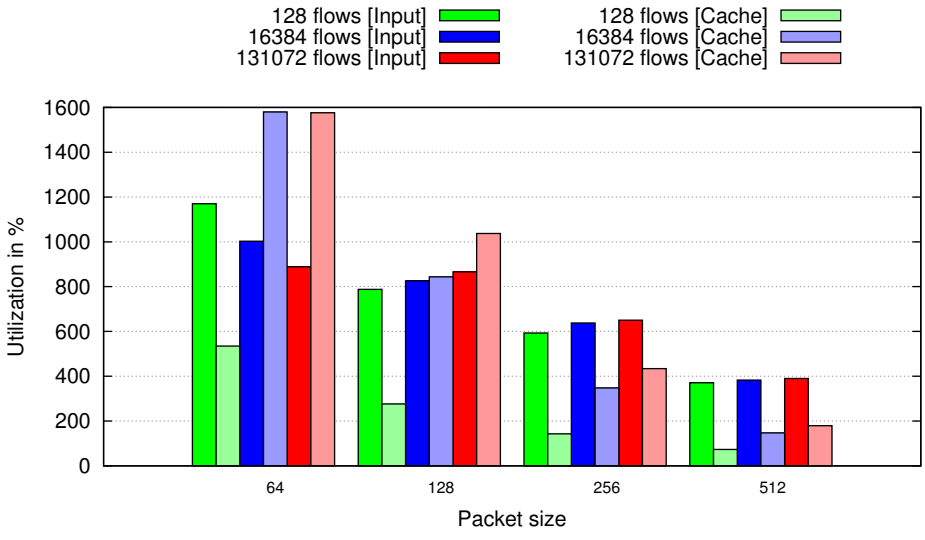


Figure 5.9: Full Packet Processing CPU Utilization.

There are several important observations that can be made from these two graphs. The performance impact of the number of flows is significant, especially for short packet lengths. This is caused by a high number of updates that must be performed in the flow cache. For a smaller number of flows, the CPU can keep a larger portion of the flow records cached in a faster memory. As the number of flows increases, the memory is accessed more at random, which causes more cache misses and eventually a performance decrease. Figure 5.9 clearly shows, that the utilisation of the flow cache thread rises with the number of flows for all packet lengths.

Performance of the second card is always higher than that of the first card. We attribute this to the device driver, which allocates packet buffers without considering the heterogeneity of the NUMA architecture. In our case, the first card and the corresponding flow exporter always access the RAM through the QPI bus using memory subsystem of the other CPU, which decreases the performance.

Although it seems that the performance is decreasing for longer packets, this depends on the point of view. The number of bytes per second is actually increasing for longer packets. Since fewer packets need to be processed to achieve the same throughput, the CPU utilisation decreases. However, more data are processed, which requires higher memory controller throughput. Therefore, for longer packets, the performance is not hindered by insufficient CPU frequency, but by high memory bus utilisation.

Hardware Accelerated Performance

Secondly, we have measured the performance using hardware acceleration. Two different acceleration methods have been used. The first method is to set up the COMBO-80G cards to trim the packets to 64 bytes. This allows keeping the processing speed the same for all packet lengths. The second method uses internal parser of the NIC and sends only a predefined data structure, called Unified Header, with parsed information to the software. The main advantage is that the flow exporter does not have to parse the packet headers and therefore saves some CPU time. The disadvantage of both methods is that the flow exporter cannot perform any payload dependent analysis.

Figure 5.10 shows the performance comparison of the processing of full packets, trimmed packets and unified headers. We plot the graphs for 16 384 flows per interface, but the results are very similar for other flow counts. Note the throughput of PCI Express bus. Using the Unified Headers decreases the number of bytes that must be transferred for each packet. Therefore the throughput is higher even for the shortest packets.

Using trimmed packets does not achieve any improvements for the shortest packets. However, it allows transferring almost full Ethernet speed on 128 B packets to software. Full line rate transfer can be achieved for 256 B packets using packet trimming or using Unified Headers even for 128 B packets.

The CPU utilisation, shown in Figure 5.11 should be the same for full and trimmed packets for same packet rate. The difference is caused by the fact that higher packet rate can be achieved using trimming. Utilizing the Unified Headers brings the advantage of easier data processing. Therefore, the input thread is always less utilised for the Unified Headers than for trimmed packets. Using the Unified Headers also brings higher performance. The parsing of more complex L3 and L4 headers also causes memory accesses. When it is reduced, more bandwidth is left for the flow cache, which increases the overall performance.

Impact of CPU Choice

We have also investigated the impact of the choice of CPU on the packet processing performance. We performed the same tests on the same server with different CPUs. We chose E5-2620 v1 CPUs, which represents a cheaper and therefore slower alternative. Each CPU features six

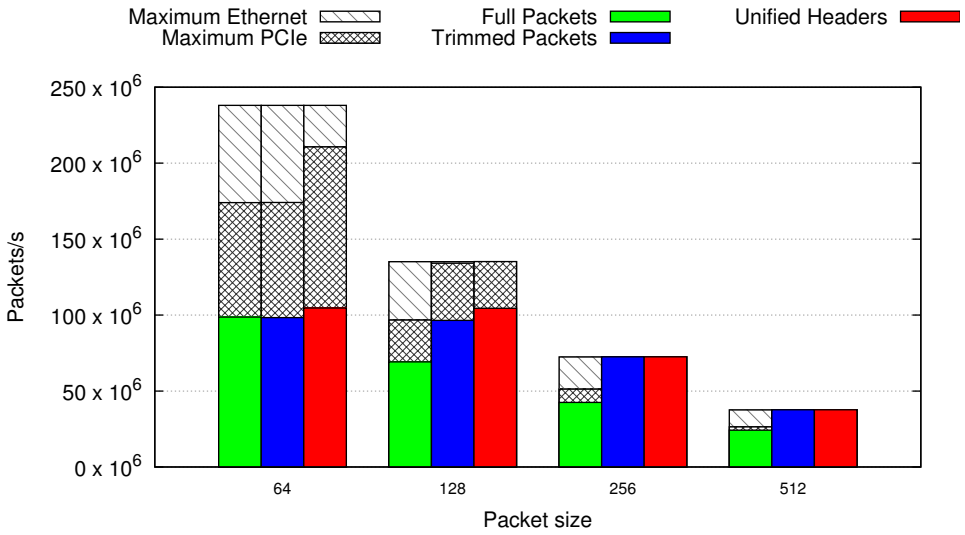


Figure 5.10: Packet Processing Performance Comparison in Packets/s for 16,384 Flows per Interface.

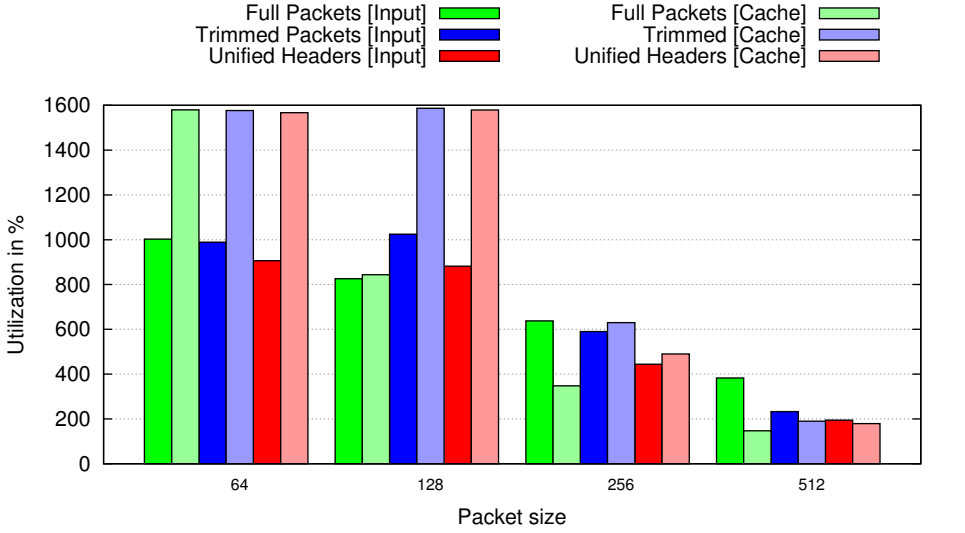


Figure 5.11: Packet Processing Comparison Using CPU Utilization for 16,384 Flows per Interface.

physical cores (12 with hyperthreading), 2.0 GHz operating frequency (2.5 GHz in Turbo mode) and 15 MB of cache. Maximum TDP of each unit is 95 W. The memory controller has a throughput of 42.6 GB/s.

Figure 5.12 shows a comparison of the performance for the trimmed packets method. The darker colours are used for the faster CPU. The difference is bigger for the smaller number of flows and is reduced with growing utilisation of the flow cache. On 64 B packets the performance drop using the slower CPU is 29 % for 128 flows, 23 % for 16,384 flows, and 21 % for 131,072 flows.

We assume that the main difference in the performance is caused by the number of CPU cores. Since there are eight independent DMA channels for the data transfer, the best performance is achieved by eight threads processing the data. However, when only six cores are available, the threads must share the cores and the overhead of switching the threads increases. The difference in the frequency helps mostly for parsing the data; therefore it is useful mainly on full

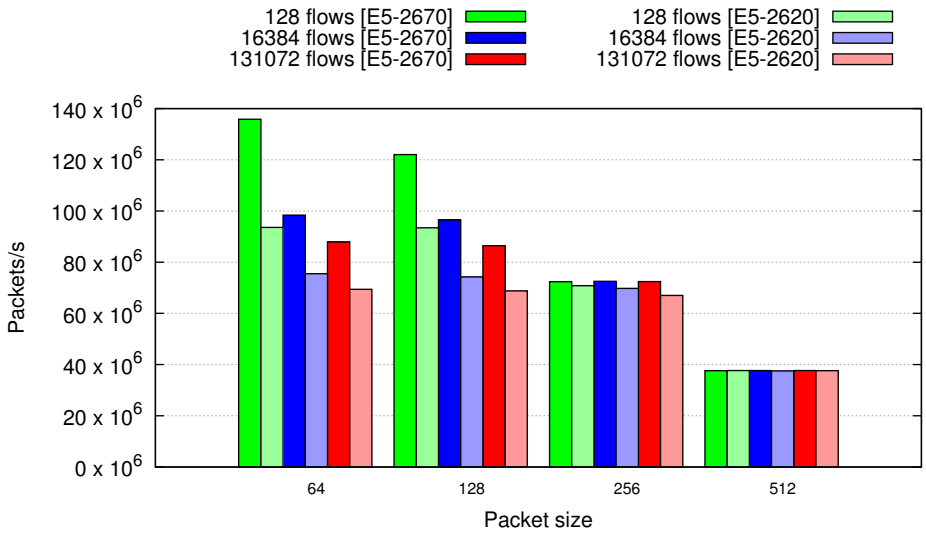


Figure 5.12: Trimmed Packets Processing Performance Comparison for Two Different CPUs.

packets. Another bottleneck is caused by the memory controller since the flow cache updates generate lots of random memory accesses. Using a CPU with wider memory bandwidth helps to alleviate this issue.

5.4.4 Conclusions

The demand for high-density network flow monitoring solutions is increasing. We have built a 2U standard rack server with two COMBO-80G cards. Each card has two 40 G interfaces and can be used in 8×10 G mode. Therefore, the server is theoretically capable of monitoring sixteen 10 G lines, 160 Gbps in total.

We have used a packet generator to fabricate a traffic of different properties and used this traffic to test the capabilities of the monitoring solution. We have worked with different packet lengths and different numbers of active network flows.

Firstly, we have measured the monitoring throughput on full packets, which is a setup that can be achieved using any commodity card with a timestamping unit and an effective distribution among CPU cores. We have shown that the monitoring performance does not achieve the highest possible rate for short packets. However, the maximum rate allowed by the PCI Express bus is achievable for longer packets.

There are several caveats that need to be kept in mind while working with NUMA architecture. Each NIC is connected to one CPU and should store packets in the memory of that CPU. This needs to be enforced by the driver of the NIC. Consequently, the flow exporter for the NIC should also run on the same CPU and work with the corresponding memory. The drivers that we use are not NUMA-aware and the deficiency is clear from the results.

Secondly, we have shown the impact of hardware acceleration on the flow monitoring. Using a packet trimming ensures that the packet rate that can be achieved for short packets applies for longer packets. This technique allows us to monitor full speed 16×10 G Ethernet for 256 B packets. For comparison, the average packet length of our organisation's border lines is over 800 bytes. When the packet parsing is performed by the NIC and the information from packet headers are passed using the Unified Headers, the performance is increased even more. The CPU utilisation is also lower since the packet header parsing is a CPU intensive task. However, this approach trades high performance for monitoring flexibility.

The purpose of the last test was to show the difference that can be achieved by using faster CPUs. We have shown that the performance on short packets can rise by more than 30 % when

using faster CPUs with higher memory bandwidth. We conclude that the monitoring system can achieve even higher performance by utilising better CPUs. However, a cost to performance ratio is also important for a production use.

More attention needs to be paid to packet loss management. While we aim to achieve a lossless flow monitoring, packets are lost in extreme scenarios nonetheless. In our case, the packets are lost by the NIC at the input when the input buffers are full and DMA transactions are stalling. Therefore, by blocking a single DMA channel packets are lost for all channels. This is clearly suboptimal since the other cores cannot process more packets while their RX queues are not receiving packets as well. An improved design for the NIC, where the packets will be dropped for each DMA channel separately, should be implemented in the future.

There are several improvements that can be made to achieve higher performance. We can buy better CPUs, but the budget for monitoring systems is often limited. The driver for the NICs should be made NUMA-aware to avoid costly memory accesses using QPI bus. Moreover, the current driver allows reading only a single packet at a time. Therefore, processing packets in batches could be used to achieve better performance. We have also shown that features like packet trimming can significantly improve the performance of flow monitoring systems. If the next generation commodity cards should be used for the network flow monitoring, such features can turn out to be crucial for handling large volumes of data. The performance of the PCI Express bus can be doubled by using 16 lanes, which is an approach that can be expected to be used in the future.

5.5 Summary

This chapter has discussed the performance of flow monitoring systems. We have explained different methods of measuring the overall performance and have shown that it is important to choose the one that provides more appropriate results. Different factors that have an impact on the flow monitoring were considered as well. We have shown that the used hardware, system settings, packet capture framework, settings of the flow exporter, and the traffic mix have a considerable impact on the results of the measurement. CPU processing power and memory controller throughput are the most obvious bottlenecks for flow monitoring. We have shortly discussed ways to detect these bottlenecks and decide whether the memory footprint or CPU cycles need to be optimised.

The performance of flow monitoring significantly depends on the performance of the underlying packet capture framework. We have presented the evolution of the state-of-the-art packet capture frameworks. Linux NAPI can be used for packet capture on gigabit networks, however, to achieve full packet capture on 10 Gbps and faster networks, specialised NIC drivers together with kernel network stack bypass must be used. Zero-copy approach and receive side scaling are a necessity to achieve these speeds. The buffers to which the NIC delivers packets through DMA transfers are mapped directly to the user-space. Interrupts are usually disabled at high speeds and extensive polling is performed by the receiving application. When the data needs to be shared between multiple applications, the RX queues are managed by the kernel driver. In this case, the provided API must enable reception of multiple packets at once, a technique called batch or burst packet processing, to avoid unnecessary context switches between kernel and user-space. In case only a single user-space application is processing the packets, such as in the DPDK framework, the RX queues are managed directly by user-space, which increases the performance due to the lesser number of context switches.

Although packet capture is an important part of the flow monitoring process, many optimisation techniques can be applied to the flow monitoring as well. We have discussed the possibility of hardware acceleration using specialised FPGA-based NICs that allow offloading of part of the flow processing to the NIC itself. We have shown that depending on the capabilities

of the NIC, different levels of offloading can be achieved. However, the flow exporter must be aware of these capabilities and actively cooperate with the NIC. Multiple optimisations can be used in the design of the flow exporter software as well. We have argued that multithreading and NUMA awareness are key to flow monitoring performance, although special care needs to be taken to configure the setup correctly. Other optimisation techniques such as flow cache design, per-flow expiration timeouts, delayed packet parsing, and the use of biflows have been considered as well.

Finally, we build a high-density flow monitoring system capable of processing 16x10 Gbps. The system utilised two FPGA-based NICs that allowed us to test the offloading of packet processing. We have shown that this technique allows processing significantly higher amount of traffic. The number of concurrent flows that needed to be kept in a flow cache has had a considerable impact on the measured performance. Moreover, we also demonstrated the importance of having powerful enough CPU by repeating the tests with two different CPUs. We believe that by utilising more of the proposed optimisation techniques, we should be able to achieve even higher throughput of the system, possibly reaching even 200 Gbps rate offered by the latest NICs available on the market.

Chapter 6

Measurement of Encrypted Traffic

The application flow monitoring relies heavily on the ability to observe and process data from the application layer. However, users and companies are becoming more privacy conscious, and the use of encryption is steadily increasing. This chapter presents an overview of current approaches for the classification and analysis of encrypted traffic. The contribution of our work is four-fold. First, we describe several of the most widely used encryption protocols to show their packet structure and standard behaviour in a network. This information forms the basis of all classification protocols which use either a specific packet structure or a communication pattern to identify the protocol. Second, we investigate what information is provided by these encryption protocols. Most protocols negotiate encryption algorithms in clear-text, these data can be monitored, e.g., to reveal the use of weak ciphers. Third, we describe how the structure of encryption protocols can be used to detect these protocols in a network. Traffic classification algorithms using such information are presented, and we describe several open-source tools which implement these algorithms. Fourth, we provide an extensive survey of behaviour-based methods for encrypted traffic classification. We show that surprisingly detailed information can be obtained using these methods. In specific cases, even the content of the encrypted connection can be established.

The article included in this chapter is [A3].

The organisation of this chapter is as follows:

- *Section 6.1 provides motivation for this chapter and defines its goals.*
- *Section 6.2 explains the principles of several widely used encryption protocols.*
- *Section 6.3 describes what information can be obtained from encrypted traffic using specific knowledge of the encryption protocols.*
- *Section 6.4 describes the traffic classification taxonomy used in this chapter.*
- *Section 6.5 describes traffic classification methods based on specific knowledge of encryption protocols.*
- *Section 6.6 surveys papers on the classification of encrypted traffic using statistical and behavioural methods.*
- *Section 6.7 concludes the chapter and provides directions for future research.*

6.1 Motivation and Goals

Network visibility is becoming a necessity in current networks. Security, traffic provisioning, and failure detection are the prime reasons to deploy traffic measurement. Yet, measurement has other uses, and new ones are still being discovered. For instance, application performance can be measured using the data from the application layer. Information about certain applications can also be used to detect attacks and intrusions on the application level. In contrast to this, the need for protection of transmitted data and user privacy is rapidly increasing. It is for this reason that the encryption of transmitted data is increasingly used. The ratio of encrypted traffic has recently been rising steeply as common Internet services become protected [228]. This change poses a challenge to currently used methods for traffic measurement, for which the identification and analysis of network traffic become increasingly difficult.

Before any further analysis of encrypted network traffic can be done, the traffic needs to be identified. Statistical and behaviour-based application identification methods are less affected by encryption than deep packet inspection methods. Therefore, a lot of attention has been given to these methods, which are also considered to be privacy-conscious. However, information can even be extracted from encrypted connections, mainly from the session's initiation.

The goal of this chapter is to provide a comprehensive overview of methods for classifying and analysing encrypted traffic. To the best of our knowledge, this is the first work which comprehensively summarises available approaches for encrypted traffic classification and analysis. Although there has been a lot of research on traffic classification in the past decade [87, 86, 229, 230, 88], most of the existing surveys do not explicitly consider research which targets encrypted traffic. A recent survey by Cao et al. [231] describes the fundamentals of encrypted traffic classification. It briefly introduces recent advances and challenges in this field. However, the survey covers only a few methods of traffic classification and does not provide any comparison or assessment of these methods. Moreover, while traffic classification is an important part of network monitoring, there are other methods for analysing encrypted traffic which need to be taken into consideration.

To achieve this goal, we have produced a survey of methods for classifying and analysing encrypted traffic in journals, conference papers, proceedings of specialised workshops, and technical reports. We studied works presented in selected computer science journals, mainly the Communications Surveys & Tutorials, Computer Networks, International Journal of Network Management, and Transactions on Network and Service Management. We also surveyed international conferences such as IMC, PAM, CISDA, CNSM, IM, and NOMS over the period 2005-2014. Other methods were found in references provided by the surveyed papers and in papers that referenced them.

Over the last decade, many statistical and machine learning algorithms have been applied to the problem of traffic classification. However, authors use different methodological datasets to evaluate their methods, and the results are therefore not directly comparable. Most of the methods use supervised or semi-supervised machine learning algorithms to classify flows and even determine the application protocol of the flow. Most methods target encryption protocols such as SSH, SSL/TLS, and encrypted BitTorrent. To describe and categorise the classification methods, we use the taxonomy of Khalife et al. [232].

6.2 A Description of Encryption Protocols

This section provides a description of chosen encryption protocols. We selected the most widely used protocols to demonstrate the basic principles and show different approaches for secure data transport. Our choice of encryption protocols was also influenced by protocols which are the most commonly used in research of encrypted traffic classification. We shall describe IPsec,

TLS, SSH, BitTorrent, and Skype protocols in this section. All these protocols provide confidentiality using encryption, and they usually offer authentication of communication peers, data integrity, replay protection, and non-repudiation. We shall describe the use of these protocols, their fundamental properties, the structure of encrypted packets, and the type of exchanged data. The information provided in this section will serve as the basis for rest of this chapter.

Almost all the presented encryption protocols can be divided into two main phases: the initialisation of the connection and the transport of encrypted data. The first phase can be further divided to an initial handshake, authentication, and a shared secret establishment. During the first phase, algorithm capabilities are usually exchanged, communication parties are authenticated, and secret keys are established. These keys are then used for encrypting transferred data in the second phase. This general protocol scheme is depicted in Figure 6.1, which, with minor modifications, can be applied to almost any protocol providing confidential data transfer.

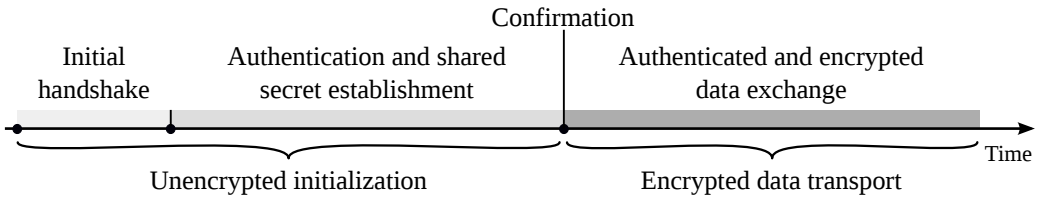


Figure 6.1: A General Scheme of Network Security Protocols.

The protocols presented in this section are listed in order of their position in the ISO/OSI reference model [233]. IPsec protocol suite, which operates on the network layer is described first, followed by TLS and SSH protocols on the presentation layer. BitTorrent and Skype protocols represent the application layer, and they implement their own protocols for secure data transmission.

6.2.1 Internet Protocol Security

Internet Protocol Security (IPsec) is a framework of open standards for ensuring authentication, encryption, and data integrity on the network layer. Due to its location on the network layer, IPsec can protect both the data within the packet and also L3 information (e.g., IP addresses) in each packet [234]. The main advantage of using IPsec is securing the network connection without the necessity of modifying any application on the clients or servers. However, it provides less control and flexibility for protecting specific applications.

IPsec follows the general scheme depicted in Figure 6.1. The first phase is represented by the Internet Key Exchange Version 2 (IKEv2) protocol [235]. IPsec uses a UDP protocol on the port 500 through which all messages covering the initial handshake, the authentication, and the shared secret establishment run. Two protocols could be used in the second phase of IPsec: Authentication Header (AH) and Encapsulating Security Payload (ESP). In the initial version of IPsec, the ESP protocol providing data confidentiality did not include authentication, so ESP and AH were used together. Nowadays, the current version of ESP contains authentication, and AH has become less significant, although it is still used to authenticate portions of packets that ESP cannot manage.

The ESP protocol is the main protocol of IPsec. It provides data confidentiality, origin authentication, connectionless integrity, an anti-replay service, and limited traffic flow confidentiality [236]. ESP adds a header and a trailer to each transferred packet, see Figures 6.2 and 6.3, placed according to the transport mode used. The ESP and AH protocols can operate in two modes: *transport* and *tunnel*. In the *tunnel* mode, a new IP header is created for each packet with

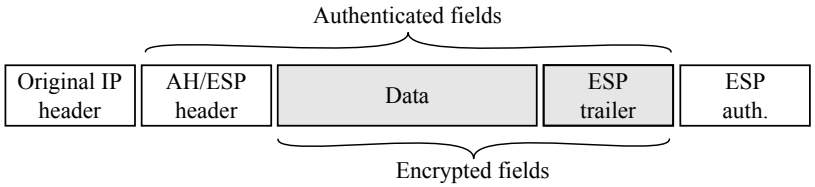


Figure 6.2: The IPsec Packet Structure in the Transport Mode.

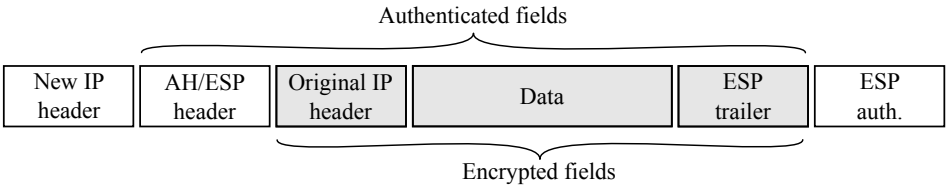


Figure 6.3: The IPsec Packet Structure in the Tunnel Mode.

endpoints of the tunnel as the source and destination addresses; the original IP header is used in the *transport* mode.

6.2.2 Transport Layer Security

Transport Layer Security (TLS) [237] is based on the Secure Sockets Layer version 3 (SSLv3) protocol [238] and provides transport level security directly on top of the TCP protocol. Specifically, it provides confidentiality, data integrity, non-repudiation, replay protection, and authentication through digital certificates. The TLS is currently one of the most common protocols for securing network communication. It is mainly used for securing HTTP, FTP, SMTP sessions, as well as for Virtual Private Networks or Voice over Internet Protocol (VoIP). The protocol design is layered and consists of different sub-protocols, as well as configurable and replaceable cryptographic algorithms [239].

The main part of TLS is the Record Protocol [237], which acts as an envelope for application data as well as TLS messages. In the case of the application data, the Record Protocol is responsible for dividing the data into optionally compressed fragments. The addition of fragments to the record is complemented by Message Authentication Code (MAC). For more details, see Figure 6.4. Depending on the selected security algorithms, a fragment and MAC are encrypted together and sent as one TLS record. A packet may contain more than one record to avoid sending multiple short packets.

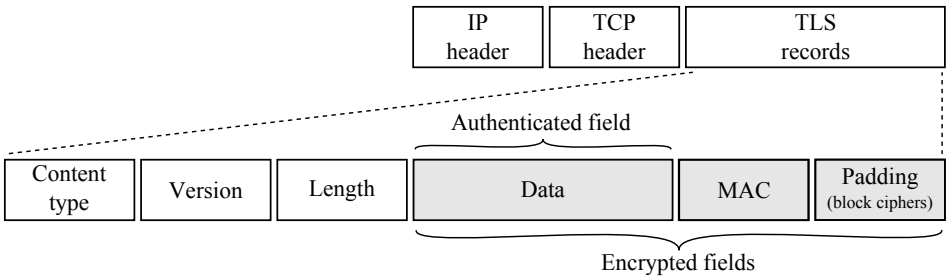


Figure 6.4: The TLS Record Packet Format.

During the first phase of a TLS connection, communicating parties are usually authenticated (more often we can see only server authentication) using an X.509 certificate chain [240], as shown in the general scheme in Figure 6.1. Alternatively, a previous connection can be resumed

without authentication. TLS messages exchanged during this phase are unencrypted and do not contain MAC until the shared keys are established and confirmed. In the second phase, these keys are used directly by the Record Protocol, which is based on the selected algorithms ensuring communication security.

6.2.3 Secure Shell Protocol

In a similar fashion to the TLS protocol, the Secure Shell (SSH) protocol [241] exists as a separate application on top of TCP. This protocol uses a client-server model where the server usually listens to the TCP port 22. SSH was originally designed to provide remote login access to replace unsecured Telnet connections. Nowadays, it can be used not only for remote login and a shell, but also to allow file transfers using the associated SSH File Transfer Protocol (SFTP) [242] and a Secure Copy (SCP) [243] protocol, or by Virtual Private Networks (VPN). The SSH protocol provides user and server authentication, data confidentiality and integrity, and optional compression.

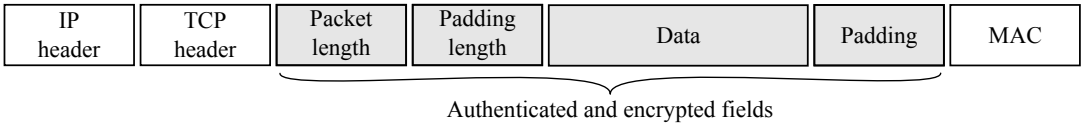


Figure 6.5: The SSH Protocol Packet Format.

SSH consists of three protocols, of which the most important is the Transport Layer Protocol, which provides the establishment of the whole connection and its management. It defines the SSH packet structure, which is depicted in Figure 6.5. The MAC code is computed on a plaintext payload together with the packet length, the padding, and a continuously incremented sequence number not present in the packet itself. Other SSH protocols are the User Authentication Protocol and the Connection Protocol for multiplexing multiple logical communication channels [244].

Each SSH connection passes through the same phases which were depicted in Figure 6.1. In the first phase, a TCP connection is established, and information about preferred algorithms is exchanged. During authentication, the server sends its public key which must be verified by the client (using a certification authority or manually through a different channel). The shared keys are subsequently established and confirmed. All following packets are then encrypted and authenticated.

6.2.4 BitTorrent

BitTorrent [245] is an application protocol based on the principle of peer-to-peer network communication for sharing large amounts of data over the Internet. Originally, the protocol did not ensure any type of network communication security. Once the popularity of this protocol increased, some Internet Service Providers (ISP) started to limit this type of traffic. As a response to this, the Message Stream Encryption (MSE) algorithm [246], also known as Protocol Encryption (PE), was introduced. It serves as an obfuscation algorithm to make BitTorrent traffic identification more difficult. In addition to obfuscation, the mechanism also ensures some level of confidentiality and authentication for communicating peers.

The MSE protocol specification [246] describes MSE as a transparent wrapper for bidirectional data streams over TCP which prevents passive eavesdropping and thus protocol content identification. MSE is also designed to provide limited protection against active man-in-the-middle attacks and port scanning by requiring a weak shared secret to complete the handshake.

The major design goal was payload and protocol obfuscation, not peer authentication or data integrity. Thus, it does not offer protection against adversaries which already know the necessary data to establish connections (that is the IP/port/shared secret/payload protocol).

The first general phase of the MSE protocol follows a TCP three-way handshake and starts with a newly generated Diffie-Hellman (DH) public key exchange (together with random data padding for better obfuscation). The shared key is computed by the DH key and combined with hashed information about the requested data which acts as a pre-shared secret. The packet's payload is completely encrypted by an RC4 stream cipher after successfully confirming the shared key.

6.2.5 Skype

Skype [247] is a peer-to-peer based VoIP application providing not only video chat and voice calls, but also an instant messaging and file exchange service. As the protocol used is not publicly known, it is not possible to accurately describe its specific details. The main reason for this is network data obfuscation to make the detection of Skype traffic more difficult, which is similar to the BitTorrent protocol.

The Skype protocol operates over both UDP and TCP protocols depending on network characteristics. If the network has no restrictions, the application usually sends data traffic over UDP and the signalling traffic over TCP [248]. If UDP cannot be used (for example a firewall prevents users from using such a protocol), Skype sends both the signalling and the data traffic over TCP. When TCP is used, the connection is usually established over port 80 or 443, which masks Skype traffic as standard web traffic.

Skype uses the TLS protocol over the TCP port 443 and a proprietary protocol over port 80 [248] for securing and obfuscating generated traffic of each communicating peer. TLS is also used in communication with other Voice over IP solutions, where it is used for protecting Session Initiation Protocol (SIP) messages [249]. Skype uses a proprietary protocol for communication over UDP. To offer a reliable connection, UDP packets contain an unencrypted header with a frame ID number and function mode fields. The encryption in UDP connections is used only for obfuscation and not for confidentiality; therefore, there is no generated shared secret, only a proprietary key expansion algorithm. UDP connections do not follow the general scheme in Figure 6.1 because encrypted data are directly transferred without an initialisation phase.

6.3 Information Extraction from Encrypted Traffic

Network monitoring is one of the main pillars of network security. If the appropriate data are collected, it is possible to detect network attacks and trace attackers, detect security policy violations and monitor network applications performance. If encrypted traffic is used, the possibility of information extraction is significantly limited. Nevertheless, it is possible to obtain some information from this traffic, primarily from the unencrypted initialisation phase, but also from the encrypted transport phase. This section begins with a description of the initialisation phase, which is then followed by a description of methods which use traffic feature analysis to gain information from the transport phase.

6.3.1 The Unencrypted Initialization Phase

Almost all network protocols ensure secure data transfer by means of encryption containing an unencrypted initialisation phase, as depicted in Figure 6.1. Because the data exchanged at this stage are not encrypted, they can be easily extracted and used for monitoring of network traffic. Generally, two types of information common to most protocols can be extracted during this phase. The first type covers the connection itself, and its properties exchanged in the initial

handshake. The second type covers communicating peers' identifiers which are exchanged in the authentication phase.

During the initial handshake, parameters of a connection are negotiated, such as which cipher suite and protocol version is used. This dynamic setting of the connection properties enables backward compatibility for different versions of software or is used to set a different level of security based on established security policies. Some examples of this are data authentication and compression in addition to the encryption itself. The list of possible identifications, with references to algorithm specification for IPsec, TLS, SSH, and other protocols, can be found in IANA Protocol Registries [250]. All of this information can be used for proper connection characterisation and correct parsing of other packets in the rest of the connection.

One interesting use of the information from the initial handshake is presented by client fingerprinting based on the provided cipher-suites. A large amount of cipher-suites types exists, which usually are not all implemented by the client's applications. Therefore, each application specifies the supported cipher-suites and also the order of their preference during the initial handshake. This makes it possible to passively distinguish specific operating systems, web browsers, and other applications, together with their versions, based only on the cipher-suites which they use. An example of such client fingerprinting, based on the SSL/TLS initial handshake, is presented by applications from SSL Labs [251] and p0f module [252].

The authentication phase represents the second source of information which can be easily extracted from secured network traffic. Unique identifiers of one or both communicating peers, optionally supplemented by additional data about them, are exchanged during this phase. For example, in the authentication phase of the SSH protocol, the server sends its public key to the user. The user must validate the key and verify that the server knows the appropriate private key [241]. Since this information is almost unique for each SSH server, it is possible to detect server changes or man-in-the-middle attacks on them by passive network traffic monitoring.

A very good example of extracting information from communication peers is demonstrated by monitoring the authentication phase of the SSL/TLS protocol. The server, and optionally even the client, send their X.509 certificates [240] to each other to verify their identities in this phase. This certificate contains a public key signed by the certification authority which is supplemented with information about the peer and issuer. A more detailed content of such certificates is shown in Figure 6.6.

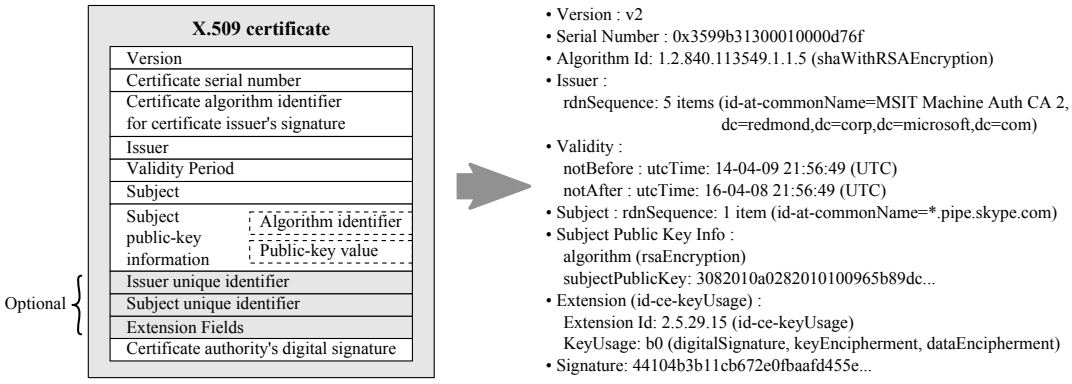


Figure 6.6: Example of Skype X.509 Certificate.

Passive monitoring of certificates enables not only the identification of communicating peers but also the ability to check whether these certificates are valid and contain proper security algorithms to fulfil local security policies. SSL/TLS certificate properties were studied by Holz et al. [253] who revealed a great number of invalid certificates some of which were shared between a large number of hosts. The work of Holz et al. was followed by Durumeric et al. [254] who

focused mainly on assessing certification authorities. Certificate monitoring can also be used to detect malicious software trying to hide its activities by connecting with their command and controls centres using the SSL/TLS protocol [255].

Even though information extraction from encrypted traffic is not a computationally intensive process, it can provide valuable information. For example, extracting the Server Name Indication (SNI) [256] can be used by a home router's firewall to filter traffic. The unencrypted initialisation phase is often used to recognise encrypted traffic, and the authentication information might be utilised to detect and prevent man-in-the-middle attacks on a network-wide level.

6.3.2 The Encrypted Data Transport Phase

Information about network traffic can be extracted from encrypted data which is transported between communicating parties. Packets exchanged during the transport phase usually contain only information about the packet itself, such as the length and the authentication field which are not useful for monitoring network traffic. Nevertheless, two methods to obtain more suitable data do exist.

The first method uses direct traffic decryption, which is possible to perform only if the shared secret of the connection is known. Therefore, decryption can be used in networks on the servers' side where organisations know the private key of the connected server. However, such decryption would be impossible if algorithms which ensure forward secrecy [257] were used.

The second method is based on the extraction of traffic features. An example of such an analysis is presented by Miller et al. [258] who monitor the size of TLS encrypted packet sequences. Based on their data, together with various predictive models, they are able to identify a specific web page and deduce its content even though the traffic is encrypted. A similar approach was also used by Koch and Rodosek [259] for analysing SSH traffic. Another example of using traffic features is the work by Hellemons et al. [260], which focused on intrusion detection in SSH traffic. Encrypted traffic features could also be used for classifying encrypted traffic, which is described in Section 6.6.

6.4 A Taxonomy for Traffic Classification Methods

The first step in analysing network traffic is identifying the type of the measured traffic. Network traffic classification methods are used for this purpose based on the knowledge of the protocol packet structure, communication patterns, or a combination of both.The recognition of the TLS protocol, described in Section 6.2, may be seen as an example of this. This protocol can be identified based on the knowledge of the packet structure, especially the unencrypted packet parts such as the content type, version, and length. Similarly, the protocol can be identified by analysing its behaviour, e.g., the knowledge of a number and an approximate size of packets sent during the unencrypted initialisation phase.

To present the current state of research on encrypted traffic classification in a comprehensive manner, we use a taxonomy of classification methods. We choose the multilevel taxonomy by Khalife et al. [232], which provides a detailed categorisation of traffic classification methods. This taxonomy is uniquely descriptive and allows us to efficiently categorise all our surveyed classification methods. Figure 6.7 shows an overview of the taxonomy levels. On the topmost level, the authors distinguish between classification input, technique, and output. The input determines the traffic's characteristics, which are used for classification (e.g., packets or flows). The technique describes the core of the classification method, which may be, among others, payload inspection, a statistical method, or a machine learning method. The output then describes how the traffic objects (packets or flows) map to traffic classes (application types or application protocols). The traffic classes are of different granularity. Some methods allow identification of

application protocols (e.g., HTTP), and some are more fine-grained to detect, for example, a Google search or a Facebook chat.

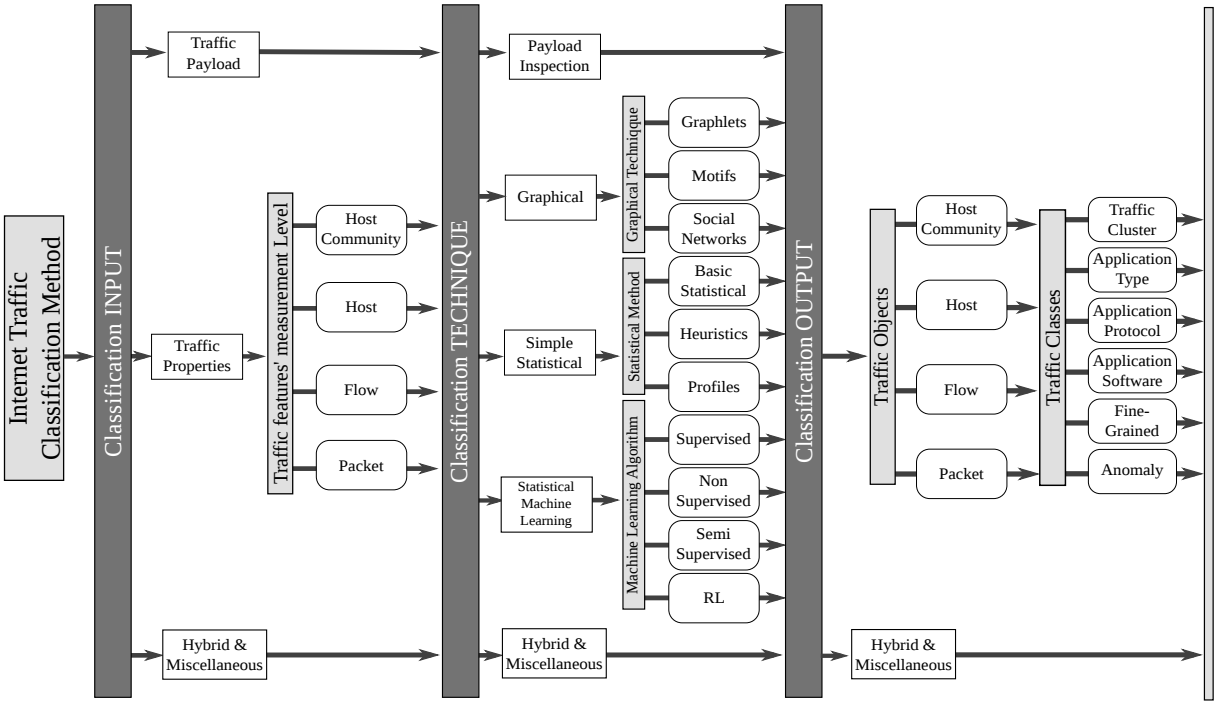


Figure 6.7: A Multilevel Taxonomy of Traffic Classification Methods [232].

Classification Input		
Traffic Payload		use of application data
Traffic Properties (Measurement Level)	Host Community	graph metrics (diameter, connection degree)
	Host	number of connections, opened ports
	Flow	flow size, flow duration
	Packet	packet sizes, inter-arrival times
Hybrid & Miscellaneous		combination of inputs, external knowledge

Table 6.1: Classification Input Level.

Tables 6.1, 6.2, and 6.3 provide examples for each of the input, technique, and output category. The input and technique tables have the most general categories in the left column, some of which are divided into more specific subcategories. The classification output table is divided horizontally into traffic objects and classes. The objects describe what is being classified, in other words, whether it is each packet, whole flow or a host. Traffic classes describe the type of classification being performed by a specific algorithm.

6.5 Payload-Based Traffic Classification Techniques for Encrypted Traffic

Almost every network traffic encryption protocol has a specific packet format that differs from others, as was described in Section 6.2. Thus, with knowledge of these formats, it is possible to distinguish and identify individual protocols by inspecting the packet payload. It is for this purpose that string or regular expression matching algorithms are used together with specific protocol patterns. Some examples of contemporary classification tools which use payload inspection are discussed in more detail in the first part of this section. The second part presents

Classification Technique		
Payload Inspection		DPI, examination of first N bytes
Graphical Techniques	Graphlets	relationship between features (ports, addresses)
	Motifs	patterns of communication
	Social Networks	graph of communication
Statistical Method	Basic Statistical	probability density functions of e.g., packet sizes
	Heuristics	port-based classification
	Profiles	host profiling, usage of packet sizes and direction
Machine Learning Algorithm	Supervised	Hidden Markov Models, Naive Bayes, k-nearest neighbour, support vector machine
	Non-Supervised	clustering of unlabeled traffic, k-nearest neighbour
	Semi-Supervised	clustering of mixed traffic, k-nearest neighbour
	Reinforcement Learning	-
Hybrid & Miscellaneous		combination of methods, external knowledge

Table 6.2: Classification Technique Level.

Classification Output	
Traffic Objects	
Host Community	host community is assigned a class, e.g., community of HTTP servers
Host	host is assigned a class
Flow	flow is assigned a class
Packets	packet is assigned a class
Traffic Classes	
Traffic Cluster	bulk or small transactions
Application Type	game, browsing, chat
Application Protocol	HTTP, HTTPS, FTP
Application Software	client software such as Mail client, FTP client or web browser
Fine-Grained	Skype voice call, Google search, Facebook chat
Anomaly	port scan, brute-force attack
Hybrid & Miscellaneous	combination of outputs or classes, external knowledge

Table 6.3: Classification Output Level.

current research papers which focus on comparing these tools in terms of their performance and success rate.

6.5.1 Payload-Based Classification Tools

Most network traffic classification tools aim to recognise all network protocols and not only the encrypted ones. The following examples represent the most widely used tools for classifying network traffic. Most of these tools are also able to distinguish specific network applications, mainly in unencrypted traffic. In terms of the taxonomy, these tools mostly use the *Payload Inspection* technique on the *Traffic Payload* classification input to map *Flows* to *Application Protocols*.

PACE [261] is a commercial classification library written in C, which uses pattern matching augmented by heuristics, behavioural, and statistical analysis. In addition to standard protocol and application recognition, it is able to identify obfuscated protocols such as BitTorrent or Skype. According to its website, PACE is able to identify thousands of network applications and protocols.

Cisco Network-Based Application Recognition (NBAR) [5] is another example of a commercial tool for classifying network traffic. This tool is primarily used on Cisco routers for quality and

security purposes. According to its authors, NBAR is also able to recognise stateful protocols as well as non-TCP and non-UDP IP protocols.

nDPI [98] is an open-source classifier forked from the (currently closed) project OpenDPI, which was in turn derived from PACE. nDPI analyses at most eight packets from each connection for classifying traffic, however, each packet is examined separately. If the connection contains multiple matches, then the most detailed match is returned. For encrypted traffic recognition, nDPI contains only a SSL decoder that extracts the hostname from the server certificate. By using these names, nDPI is able to identify specific network applications.

Libprotoident [100] is an open-source C library for classifying traffic. In contrast to the previous tools, Libprotoident inspects only the first four bytes of a packet payload for each direction. This makes it much faster but reduces its detection accuracy. The classification uses a combined approach of pattern matching, payload size, port numbers, and IP matching.

L7-filter [262] is an open-source classifier for Linux which is designed to classify traffic on the application layer. The initial phase of classification is based on non-payload data such as port numbers, IP protocol numbers, the number of transferred bytes, and so forth. Payload data are analysed with regular expressions during the second phase. One disadvantage of the l7-filter is that it contains a database with old patterns which was last updated in 2011.

6.5.2 A Comparison of Classification Tools

A comparison of the presented open-source tools was introduced by Finsterbusch et al. [88]. They prepared a dataset containing the traffic of 14 different network protocols such as DNS, HTTP, BitTorrent, and SMTP(S) to compare the tools. Using this dataset, they measured classification accuracy, memory usage, CPU utilisation, and the number of packets required for proper classification. The comparison showed, amongst other results, that nDPI is not able to classify the BitTorrent protocol with more than a 43 % true positive rate, although it detects SSL/TLS with 100 % accuracy. The Libprotoident tool had the highest classification accuracy on the whole analysed traffic, and was able to classify DNS, HTTP, SIP, and e-mail protocols with 100 % accuracy. Libprotoident also needs the least number of packets on average for classifying real-time traffic. Based on the comparison by Finsterbusch et al., we can say that Libprotoident is the most appropriate tool for classifying payload-based traffic, although it is more CPU intensive than the other tools.

Another comparison of the tools was carried out by Bujlow et al. [106]. They compared all of the previously presented tools. They prepared a publicly available dataset for the comparison which contained encrypted and unencrypted network traffic from 17 application protocols, 25 network applications and 34 web services. Their results show that PACE and Libprotoident are the most accurate tools. Nevertheless, the Libprotoident tool was the only classifier able to identify all the encrypted protocols which were tested. Their results were generally very similar to those in the comparison by Finsterbusch et al.

In general, the payload-based classification tools utilise regular expression matching algorithms to identify the encrypted traffic. The main difference between the tools is how much data they need to examine and whether they need both directions of the connection for the classification.

6.6 Feature-Based Traffic Classification Techniques for Encrypted Traffic

This section surveys feature-based classification methods which specialise in encrypted traffic. These methods do not require any knowledge of the encryption protocol packet structure. Instead, they use specific protocol communication patterns to classify encrypted traffic. The methods are based on the specific protocol differences described in Section 6.2, such as packet and

flow features of unencrypted initialisation or the encrypted data transport phase. This approach provides greater generalisation and allows these methods to work with new versions and types of encryption protocols without the modification of underlying algorithms.

The taxonomy specified in Section 6.4 is used to describe the individual methods. Apart from the properties defined by the taxonomy, we also provide information about datasets used for evaluating these methods. This is especially important when additional evaluation is to be performed by other research groups to verify the results of the authors. While the taxonomy provides a traffic class for classifying the application protocol, this is sometimes too coarse for our purposes. Most classification methods identify not only the encryption protocol but also the underlying encrypted application protocol. As both cases belong to the application protocol traffic class, we provide further explanation in the description of each classification method.

A slight drawback of most flow-based classifiers is in performing the classification only after the flow has expired. This prevents the possibility of a real-time response, which has led several research groups to research real-time classification using flow-features. Their methods are also included with the others in Table 6.4 and differentiated by a column describing whether the classification is done in real-time or not.

Almost 250 discriminators (flow or packet features) are identified by Moore et al. [172], which can be used to classify flow records. The authors do not propose any specific classification method themselves, however, most of the classification algorithms use a subset of these discriminators for identifying traffic. In terms of the taxonomy, the authors provide a list of traffic features which are used to infer the category of the traffic's properties for the classification input.

Most of the feature-based traffic classification methods use statistical or machine learning methods. These methods are comprehensively described in [263]. A port-based classification was used in the past to associate applications with network connections, but the accuracy of this method is decreasing with the increased use of dynamic ports and applications evading firewalls. Despite the decreased accuracy, port numbers are often utilised as one of the packet features. Furthermore, port-based classification is still quite often used to establish a ground truth for traffic classification experiments.

For easier orientation, we present the surveyed papers grouped by the class of their traffic classification algorithm. Most of the papers employ Supervised Machine Learning Methods (Section 6.6.1), Semi-Supervised Machine Learning Methods (Section 6.6.2), and Basic Statistical Methods (Section 6.6.3). The rest combine more than one method and are therefore gathered in a Hybrid Methods category (Section 6.6.4). For each surveyed paper, we specify the classification technique, classification output, and the datasets used in the description of the classification process itself. Table 6.4 in Section 6.6.5 provides a summary of the papers with all of the mentioned properties properly categorized.

6.6.1 Supervised Machine Learning Methods

Sun et al. [264] propose a hybrid method for classifying encrypted traffic. First, the SSL/TLS protocol is recognised using a signature matching method. A Naive Bayes machine learning algorithm is then applied to identify the encrypted application protocol. The authors use a combination of public and private datasets to evaluate their method. Background traffic is taken from a public dataset, BitTorrent, eDonkey, HTTP, FTP, Thunder, and GRE application protocols. The signature based recognition of SSL/TLS protocols was tested on HTTPS, TOR, ICQ, and other protocols. The identification of the underlying protocol was tested only for TOR and HTTPS protocols.

Okada et al. [265] analysed changes in flow features due to encryption. They created a training dataset with HTTP, FTP, SSH, and SMTP application protocols encrypted using PPTP and

IPsec tunnels. The authors assessed 49 flow features and analysed which of them are strongly correlated in normal and encrypted traffic. The correlated features were then used to infer functions which transform the features from normal to encrypted traffic. Therefore, standard classifiers can be used to classify the traffic after the transformation. The authors verified their method using several modifications of the Naive Bayesian classifier.

Arndt and Zincir-Heywood [266] also concentrated on the classification of encrypted traffic and compared C4.5, k-means, and Multi-Objective Genetic Algorithm (MOGA) to this end. The classification of the SSH protocol was used as an example in their study. The authors focused on the accuracy and robustness of the algorithms. Stability was tested using three different public and private datasets for teaching and evaluating the methods. Multiple different flow export settings were tested as well. Altogether, 46 flow features were used in the evaluation, however, a different subset was used by each algorithm. The C4.5 algorithm provided the best robustness, although the MOGA had a very low false positive rate when used on the same dataset it was trained on. The C4.5 was recommended for forensic analysis by law enforcement since it is applicable to a variety of networks.

Alshammari and Zincir-Heywood have published several papers [267, 268, 269, 270, 271, 272] on traffic classification using various supervised machine learning methods. They focused on recognising SSH, Skype, and in one case, Gtalk traffic using flow features without port numbers, IP addresses, or payloads. A set of 22 flow features was mostly used, although in one case the authors selected the features using genetic programming. Public datasets are used as well as a private dataset generated on a testbed network. The ground truth for public datasets was gained from port numbers, and the private dataset included the payload and was labelled with a commercial packet classification tool. The following algorithms for traffic classification were compared: AdaBoost, RIPPER, Support Vector Machine (SVM), Naive Bayes, C4.5, and Genetic Programming.

Kumano et al. [273] investigated identification of real-time applications in encrypted traffic. IPsec and PPTP encryption were applied to the web, interactive and bulk transfer flows to create an evaluation dataset. C4.5 and SVM algorithms were utilised to classify the application on these datasets. The authors then tested the accuracy of the classification using a different number of packets from the start of the flows. They measured the impact of using fewer packets on the flow features and proposed using features which show the least change to classify applications at an early stage.

6.6.2 Semi-Supervised Machine Learning Methods

Bernaille and Teixeira [274] used traffic clustering to detect applications encrypted by SSL. Their method has three steps. First, they detected SSL connections using a clustering algorithm (Gaussian Mixture Model) on packet sizes and directions of initial packets of a connection. The first three packets and 35 clusters provide good accuracy in detecting SSL traffic. After the SSL traffic is identified, the first data packets of the connections are identified. The sizes of the data packets are used by a clustering algorithm to detect an underlying application in the third step. However, the packet sizes are modified in the last step to allow for encryption overhead. The evaluation of the proposed method is done on traffic traces from live networks and a manually generated packet trace. The datasets contain HTTP, POP3, FTP, BitTorrent and eDonkey application protocols encrypted using SSL. The authors also show that using a combination of clustering and port numbers to differentiate between applications in clusters provides better results than clustering alone.

Maolini et al. [275] identify SSH traffic and determine transported application protocols, such as SCP, SFTP, and HTTP, using a k-means algorithm. Only three packet features are used: the direction, the number of bytes, and the timestamp of each packet. Their private datasets are

created from artificial traffic and contain HTTP, FTP, POP3, and SSH protocols. Control packets such as TCP handshake, retransmitted packets, and ACK-only packets are removed from the statistics as they negatively affect the precision. Authors use only first 3-7 packets to achieve a real-time identification.

Backquet et al. [276, 277] use a Multi-Objective Genetic Algorithm to select a flow feature subspace and parameters for a clustering algorithm which detects encrypted traffic. The second work employs a hierarchical k-means algorithm to increase the identification's accuracy. The authors evaluate both approaches on a private dataset captured at a university campus. SSH is used as a representative of encrypted traffic, and the ground truth is gained from the payload of the captured packets. Based on previous works, the authors argue that the feature selection and the number of clusters highly affect the overall accuracy. Therefore, the authors selected four objectives for the genetic algorithm: cluster cohesiveness, cluster separation, the number of clusters and the number of used flow features. The results show (a) that only 14 from a total of 38 flow features were used by the best-performing algorithm and (b) that using a hierarchical k-means algorithm increases the identification performance.

Bar-Yanai et al. [278] combined k-means and k-nearest neighbour clustering algorithms to construct a new, real-time classifier for encrypted traffic. The resulting classification algorithm has the light weight complexity of the k-means algorithm and accuracy of the k-nearest neighbour algorithm. They claim the method is fast, accurate, and robust in regard to encryption, asymmetric routing, and packet ordering. A labelled dataset was prepared from generated samples of the traffic, and the classification of the dataset was done using payloads of the packets. Flows shorter than 15 packets were removed from the dataset since real-time classification for such short flows is not of practical use. If available, the first 100 packets were used for the classification. The authors stress that their method is applicable in a real-time environment and tested their implementation on an ISP link. The application protocols classified are HTTP, SMTP, POP3, Skype, eDonkey, BitTorrent, Encrypted BitTorrent, RTP, and ICQ.

Zhang et al. [279] propose an improvement to the k-means clustering algorithm. By using harmonic mean to reduce the impact of random initial clustering centres, the authors are able to increase the accuracy of the k-means clustering algorithm for classifying encrypted traffic. The authors test their approach on two datasets. The first contains only data labelled SSH and non-SSH, the second contains traffic from Skype, QQ, SSH, SSL, and MSN protocols. However, the selection of the datasets and the selection of flow features used for classification were not justified in the paper.

Du and Zhang [280] used a k-means algorithm to discern traffic of three BitTorrent clients. Flow and packet header features, such as IP addresses, ports, numbers, and the length of packets, were taken into consideration. The authors generated the traffic manually and, therefore, their data sample contains only traffic from three clients. The authors highlight that the method is fast and simple and that it can be used to identify the traffic in real-time.

6.6.3 Basic-Statistical Methods

De Montigny-Leboeuf [281] described the process of identifying traffic using flows and flow features. The author showed how to derive the flow features he uses and how to apply them to the identification of the application type (e.g., interactive typing, data transfer) and block ciphers. Moreover, the author provides a list of recognition criteria for HTTP and HTTPS web browsing traffic, IMAP, POP, SMTP, SSH, Telnet, rlogin, FTP command and data, MSN chat, and TCP audio streams. A subset of 39 traffic features was used to identify each application.

Wang et al. [282] computed entropy from packet payloads and used it for classifying traffic. They differentiate between eight different traffic classes: text, picture, video, audio, Base64 encoded text, Base64 encoded image, compressed, and encrypted. The entropy is computed on

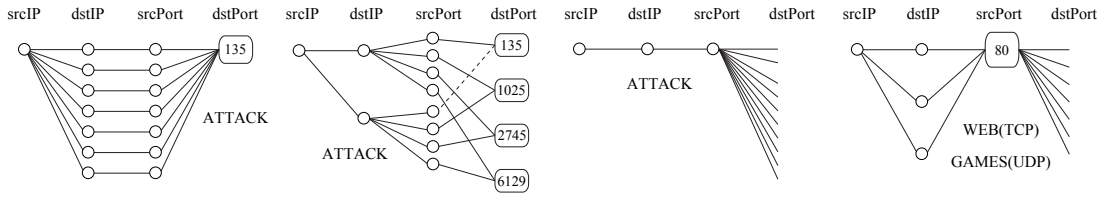


Figure 6.8: A Visual Representation of Transport-Layer Interactions for Various Applications [286].

chunks of different lengths, and the authors used a support vector machine algorithm for the feature selection. The computation of the entropy for four different chunk lengths was found to be sufficient for accuracy and performance. The most difficult task was to separate encrypted and compressed traffic. The authors provided an additional heuristic to distinguish these categories using frequencies of four-bit characters. The datasets used are manually obtained from captured network communication and processed to contain traffic from all classes.

Korczynski and Duda [283] designed a method called Statistical Protocol Identification to identify types of communication (voice call, SkypeOut, video conference, chat, file upload, file download) in Skype traffic. Nine flow features were selected using forward selection which evaluates how a given feature improves the classification performance. A private, artificially created dataset with Skype traffic was used. Other traffic with SSL, SSH, HTTP, SCP, SFTP, VoIP, BitTorrent, and other services, was used to test the robustness of the method. The authors reported high accuracy in their method, although the distinction between voice and video traffic remains a difficult problem.

Amoli and Hamalainen [284] described a Network Intrusion Detection System (NIDS) capable of detecting attacks on encrypted network traffic in real-time. The NIDS has two engines; the first one uses network change measurement to detect changes in a time-series, such as DoS and DDoS. It clusters the data to lessen the load on the second engine. The goal of the second engine is to detect the bot master behind the attack. The second engine clusters the prepared data to obtain the behaviour of the attackers and compares it with historical data. The authors used the detected anomalies to identify the botnet masters.

Korczynski and Duda [285] proposed using stochastic fingerprints based on Markov chains for identifying application traffic in SSL/TLS sessions. The method is payload-based and uses statistical information from SSL/TLS headers. The authors tested their method on twelve representative applications such as Twitter, Skype, and Dropbox. The Markov chain fingerprints are based on protocol specific distributions of packets in time. Datasets were captured from real traffic, contain only SSL/TLS traffic, and were not published. The ground truth was obtained by inspecting domain names of the SSL/TLS traffic. The authors discovered that many protocol implementations differ from the RFC specification, which required an adjustment of the fingerprints.

6.6.4 Hybrid Methods

Karagiannis et al. [286] focused on host-based classification. Their method uses only information from the network level and therefore is not affected by transport layer encryption (e.g., TLS). The authors classified the behaviour of the hosts on social, functional, and application levels without access to packet payloads or headers. Each level was classified independently, and a cross-level classification was performed afterwards. Particular applications were represented using graphlets, see Figure 6.8, which are representations of the application's behaviour. The authors then used heuristics to refine the classification. The ground truth for captured datasets

was established by using a signature-based payload classification. Without using transport layer information, only the following traffic classes were identified: web, p2p, data (FTP, database), network management (DNS, SNMP, NTP), mail (SMTP, POP, IMAP), news (NNTP), chat (IRC, AIM, MSN messenger), streaming, and gaming.

Wright et al. [287] worked on classification of traffic in encrypted tunnels. Multiple flows can be wrapped in a single flow representing the encrypted tunnel. The information from the packet headers was not applicable; therefore the authors used only packet sizes, timing, and communication direction. A k-nearest neighbour classifier was used for classification when all TCP connections in a set carried the same application protocol. When TCP connections carried different application protocols, the authors used Hidden Markov Models. The authors also demonstrated that it is possible to determine the number of flows in an encrypted tunnel. The port numbers were used to obtain a ground truth for the captured dataset. The authors argue that mislabelled data only decreased the efficiency of their classification algorithm and therefore the real accuracy would be even higher than the reported one. The classifiers were able to detect the following application protocols: HTTP, HTTPS, SMTP, AIM, FTP, SSH, and Telnet.

Koch and Rodosek [259] proposed a system for detecting interactive attacks using SSH. Packet sizes, IP addresses, and packet inter-arrival times were used to create clusters of packets which were likely to match an SSH command and its corresponding response. The SSH protocol was recognised based on the port number, and individual commands were identified from the clusters. Following this, sequences of commands were evaluated, and possible malicious sequences were reported. The system allows for the customisation of malicious sequences' definitions using a sub-goals characterisation. Each sub-goal is mapped to a malicious event, such as data gathering or system manipulation. The results from the evaluation of the proposed method show that such identification is possible.

Khakpour and Liu [288] used an entropy of packet payloads for classifying traffic. The authors showed how to compute the entropy of files and how to modify the formula for on-the-fly computation. Several entropy values were computed for each packet. CART and support vector machine (SVM) methods were used for a subsequent classification based on the computed values. Their results demonstrated that SVM methods provide comparable accuracy with less false positives. The authors argue that it is necessary to exclude application layer headers such as HTTP response or picture headers. The reason for this is that computing entropy on the headers leads to bias and misclassification of a packet. Therefore, a cut-off threshold was used to strip application headers from unknown protocols. The traffic was first classified into three categories: text, encrypted, and binary. The authors also postulated that the classification can be more fine-grained and they investigated the classification of application protocols. Then, they demonstrated that it is possible to determine an encryption algorithm with higher accuracy than random guessing, which they found surprising.

6.6.5 A Summary of Machine Learning and Statistical Encrypted Traffic Classification

We have provided a summarising overview of the feature-based traffic classification papers and methods they use in Table 6.4. Where a method belongs to multiple categories, it is not marked as a hybrid, but all the categories are listed instead. We find this approach more descriptive than using a hybrid category as defined by the taxonomy.

Most of the surveyed methods use flow or packet header features as an input for the classification techniques. The authors of [269, 272] compare the results gained by utilising packet header features and flow features. They show that using both sets of features can result in faster and more accurate classification algorithms. Nevertheless, using all the available traffic features does not necessarily lead to the best classification performance as demonstrated by the authors of [267].

Reference	Publication year	Input					Number of features	Feature selection	Technique								Output	Encrypted proto. ident.	Real-time ident.	Dataset				
			Traf. Properties								Statis.		Machine			Method(s)								
		Traf. Payload	Host Com.	Host	Flow	Packet			Payload Ins.	Graphlets	Basic	Heuristic	Sup.	Non Sup.	Semi Sup.					Public	Private	Real	Artificial	Ground truth
[281]	2005				✓	✓	⊆ 39				✓						F → AP, AT				✓	✓		port
[286]	2005		✓	✓	✓		–			✓	✓	✓					H → AT	✓			✓	✓		signature
[287]	2006				✓	✓	3						✓		✓	HMM, k-nearest neighbor	F → AP	✓			✓	✓		port
[274]	2007					✓	3								✓	Gaussian Mixture Model	P → AP	✓	✓		✓	✓		signature, known
[268]	2007				✓		22						✓			AdaBoost, RIPPER	F → AT	✓		✓	✓	–		port, known
[275]	2009					✓	3								✓	k-means	F → AP	✓	✓		✓		✓	known
[270]	2009				✓		22						✓			AdaBoost, RIPPER, SVM, Naive Bayes, C4.5	F → AP			✓	✓	✓		port, signature
[269]	2009				✓	✓	⊆ 30						✓			RIPPER, C4.5	F → AT				✓	✓	✓	signature, known
[267]	2009					✓	39	✓					✓			AdaBoost, C4.5, Genetic Programming	F → AP				✓	✓		signature, port
[276]	2009				✓		38	✓							✓	k-means, Genetic Programming	F → AP				✓	✓		signature
[271]	2010				✓	✓	22						✓			AdaBoost, C4.5, Genetic Programming	F → AP	✓			✓	✓	✓	signature, known
[278]	2010				✓		17								✓	k-nearest neighbor, k-means	F → AP		✓		✓	✓		signature, known
[264]	2010	✓			✓		6		✓				✓			Naive Bayes	F → AP	✓		✓	✓	✓		known
[259]	2010					✓	–				✓	✓					F → FG		✓	–				–
[277]	2011				✓		38	✓							✓	(hierarchical) k-means	F → AP				✓	✓		signature
[282]	2011	✓					7	✓			✓					Entropy	F → AT	✓	✓		✓		✓	known
[265]	2011				✓		49	✓					✓			Naive Bayes	F → AP	✓			✓		✓	known
[266]	2011				✓		⊆ 46	✓					✓			C4.5, k-means, MOGA	F → AP			✓	✓	✓		signature, port
[272]	2011				✓	✓	61	✓					✓			AdaBoost, C4.5, Genetic Programming	F → TP			✓	✓	✓		signature, port
[283]	2012				✓		9	✓			✓					Statistical Protocol Identification	F → AP	✓			✓		✓	known
[279]	2013				✓		22								✓	k-means	F → AP			✓	✓	✓		signature
[280]	2013				✓	✓	7								✓	k-means	F → AS				✓		✓	known
[288]	2013	✓					10	✓			✓		✓			Entropy, SVM, CART	F → AT, AP	✓			✓	✓	✓	known
[284]	2013				✓	✓	23				✓			✓		Change point detection, Clustering	F → A			–				–
[273]	2014				✓		29	✓					✓			C4.5, SVM	F → AT	✓	✓		✓		✓	known
[285]	2014	✓				✓	–				✓					Markov chains	F → AP	✓			✓	✓		signature

Output column legend:

A	Anomaly	AP	Application Protocol	AS	Application Software	T	Application Type
H	Host	F	Flow	FG	Fine-Grained	TP	Traffic Payload

Table 6.4: A Summary Table of Cited Papers and Methods They Use to Detect Encrypted Traffic.

The column *Number of features* shows how many flow or packet header features were used in each method. Some methods used different subsets of the features for different algorithms, and this is denoted by the $\subseteq$ mark. Moreover, some of the methods used a feature selection algorithm to select the best combination of features from the entire feature set. These methods are marked in the *Feature selection* column. For this case, the *Number of features* column represents the initial number of features.

Most classification algorithms are based on machine learning. The category of supervised machine learning algorithms is represented by Hidden Markov Models, RIPPER, AdaBoost, Support Vector Machines, C4.5, and Naive Bayes. Several works [268, 270, 271, 272] compare these algorithms to establish which is the best for the task of classifying traffic. The C4.5 algorithm performs the best in several cases; however, genetic programming is reported to achieve the best results in [272]. The second most common algorithm category is the semi-supervised machine learning, which is dominated by clustering algorithms. The k-means algorithm is the most frequent in this category, and the k-nearest neighbour comes second. The popularity of k-means is due to its variability, which allows it to be fine-tuned for various purposes. It is often combined with genetic algorithms to find the best setting. The authors of [282, 288] use the entropy of packet payloads to classify traffic. Using simple statistical properties of the traffic is the third most common classification method. Other methods are rarely used, mainly because they cannot learn from labelled traffic and therefore require too much effort to set up.

The SSH protocol is heavily used as a classification example. The authors of [267, 268, 269, 270, 272, 266, 276, 277, 279] test their methods for recognising SSH and non-SSH traffic. Maiolini et al. [275] take the classification one step further and identify the type of traffic encapsulated in an SSH connection. The authors of [274, 285, 264] use SSL/TLS traffic and identify underlying application protocols. Since the SSL/TLS protocol is more general and is used to encrypt various types of traffic, the complexity of identification is higher than for SSH. Another very popular protocol for identification is Skype, which is addressed by the authors of [270, 272, 283, 285].

Some of the methods focus only on identifying encrypted traffic, whereas others try to identify the underlying application protocol. The methods which perform a more thorough analysis to gain information about the application protocol are indicated in the column *Encrypted protocol identification*.

Because all methods, with the exception of [286], classify whole flows and rely mostly on flow features, they are rarely able to classify traffic in real-time. However, the authors of [278, 274, 273, 275, 282] achieved near real-time classification by extracting features of only a fixed number of packets in a flow. They argue that the first packets carry enough information for classification. Using a higher number of packets increases accuracy; therefore it is possible to strike a balance between accuracy and early identification.

The *Dataset* columns describe whether the data used to evaluate the presented methods was taken from a live network (Real) or generated by a tool (Artificial). We also identify if the datasets were publicly available (Public), were made available by the authors (Published), or kept undisclosed as they contain sensitive information (Private). If more than one dataset was used for each evaluation, we simply performed a union of the dataset descriptions. The *Ground truth* column indicates how the ground truth was obtained for each dataset. Common methods are based on port numbers or signatures, which use the packet payload. When the datasets are generated manually, the ground truth is known in advance.

The classification accuracy reported by authors of the surveyed methods depends heavily on the used datasets. All authors use their own private datasets, which are seldom published. Such methods simply cannot be compared without repeating the experiments on a common dataset. The authors of [268, 270, 272, 266, 264, 279] also used publicly available datasets which were either labelled beforehand, using payload when available, or simply labelled using port numbers. A combination of datasets is often used to test the robustness of the methods.

The surveyed methods show that a lot of effort was put into classifying encrypted traffic. We believe that there are several points that should be taken into account in any future research in this field. First, identifying encrypted traffic is not enough. The identification of the underlying protocol is the real challenge. Second, an SSL/TLS protocol should be used as the reference protocol, as it can contain much more complex traffic than the SSH protocol. Finally, the used traces should be labelled and made available to other researchers. Following these points does not limit the scope of future research; however, it simplifies the comparison of the presented approaches and allows others to verify the results more easily.

6.7 Conclusions

In this chapter, we presented an overview of current approaches for the classification and analysis of encrypted traffic. First, we selected a number of the most widely used encryption protocols and described their packet structure and standard behaviour in a network. Second, we focused on information which is provided by encryption protocols themselves. We found that the initiation phase often provides information about the protocol version, ciphers used, and the identity of at least one communicating party. Such information can be used to monitor and enforce security policies in an organisation. We also discovered that the use of information from the unencrypted parts of an encrypted connection for a network anomaly detection is only briefly investigated by researchers. Information about communicating parties can be leveraged to discern the type of encrypted traffic. For example, the list of supported cipher suites provided by a client when establishing a secure connection can help to identify the client. We believe that the use of unencrypted parts from the initiation of an encrypted connection should be explored in more detail.

Before starting the analysis of the encrypted network traffic, it is necessary to identify it. Thus, we surveyed approaches to classifying network traffic. There are payload-based methods, which use knowledge of a packets' structure, and feature-based methods, which use characteristics specific to the protocol flow. For the payload-based classification, there are several open-source traffic classifiers which can identify encrypted traffic using pattern matching. The initiation of communication often has a strictly defined structure; therefore, the patterns can be constructed for specific protocols. The main difference between various classifiers is that some of them require traffic from both directions of the communication to correctly classify the flows.

Feature-based traffic classifiers have been intensively researched over the last decade. Many statistical and machine-based learning methods have been applied to the task of traffic classification. Despite this, there are no conclusive results to show which method has the best properties. The main reason is that the results depend heavily on the datasets used and the configuration of the methods. We have applied the multilevel taxonomy of Khalife et al. [232] and categorised the existing methods. Our results show that most of the authors use private datasets, sometimes in combination with public ones. For this reason, the individual results are not directly comparable. Most of the methods use supervised or semi-supervised machine learning algorithms to classify flows and even determine the application protocol of a given flow. Most methods target encryption protocols, such as SSH, SSL/TLS, and encrypted BitTorrent, and use similar methods. However, there are also some novel works which apply innovative approaches to refine the classification up to deriving the content of the encrypted connections.

Most authors of feature-based classification methods claim that their approach is privacy sensitive as it does not require the traffic payload. However, privacy issues are much wider. In 2013, the Cyber-security Research Ethics Dialog & Strategy Workshop [289] started a discussion about the influence of cyber-security research on the privacy of Internet users. Researchers need to keep in mind that their research activities have a significant impact on infrastructure security, network neutrality, and privacy of end users.

In the past, internet protocols were not, or could not be, designed with security considerations in mind. The recent interest in privacy has motivated the IETF to reconsider this approach and discuss the privacy aspects of the protocols. Discussions held in [290] revealed that monitoring privacy issues are of great concern. This discussion resulted in a new RFC [17], where the IETF clearly states that pervasive monitoring is considered to be an attack. The document suggests that the IETF's protocols should be hardened against such monitoring. It is clear that the struggle between the demand for privacy and the need for security is still in the beginning.

Chapter 7

Next Generation Flow Monitoring

The work presented in this thesis so far aimed to improve the state of flow monitoring. We focused on including more information in flow records and on accelerating the flow monitoring for high-speed networks. The flow creation process and the structure of flow records adhered to the definitions provided in Chapters 2 and 3. The goal of this chapter is to show how the flow monitoring might evolve in the future.

We present three novel concepts that might be applied to flow monitoring to increase the amount of information carried by the flow records. All concepts require adaptation of flow data processing to a certain degree as well. Moreover, the definitions of flows given in this thesis no longer hold for these novel concepts.

The first proposed concept is called EventFlow and is based on the assumption that each user action involving the use of network usually generates more than one flow. For example, simple access to a web page might create a DNS query and an HTTP query as well. The main goal of EventFlow is to connect the flows that are part of a single action or event. We have implemented a prototype for measurement of EventFlow and provide its evaluation in this chapter.

The second concept addresses the monitoring of tunnelled traffic. Currently, each flow that represents a tunnel is split into multiple flows based on the communication in that tunnel. However, this approach usually supports only a static number of encapsulation layers. We propose another approach based on a hierarchical tree of flows.

Similarly to the monitoring of tunnelled traffic, application events can also cause a flow to be split into several smaller flows. This causes statistics relying on a number of flows as well as per-flow metrics to be inaccurate. The last concept proposes to handle application layer events separately from flow records. Each event would be associated with the original flow, however, it would not influence its creation at all.

The paper included in this chapter is [A15].

The organisation of this chapter is as follows:

- *Section 7.1 describes the design and evaluation of the EventFlow concept.*
- *Section 7.2 presents and discusses the MetaFlow concept.*
- *Section 7.3 introduces the concept of application events in flow.*
- *Section 7.4 summarizes the chapter.*

7.1 EventFlow

The application flow monitoring parses data from application headers and adds application-specific elements to flow records. This way the information from application level can be easily transferred to flow collectors, stored, and utilised together with the information about the network communication. The current approach is to treat separate application protocols individually, e.g., develop an application processing module for each monitored protocol, as shown in Chapter 3. However, connections between different protocols are lost in this scenario. For example, when a user wants to access a web page, several different flows records are created. The DNS server must be contacted to resolve the hostname of the web page to an IP address. After the basic document is loaded, the user's browser automatically loads linked content, such as images, cascading style sheets, and javascript libraries. The generated requests are recorded as flows; however, only a little relation between the flows is preserved.

Information about relations between individual flows can be useful in several scenarios. First, when an advertisement on a web page contains malware, the page can be traced using the relation and notified of the malicious content. Second, aggregates of the related flows can be created to simplify behavioural analysis of network traffic. Moreover, the analysis can use the additional information to improve its accuracy. Last, traffic classification engines can also benefit from having access to information about flow relations [291].

In this section, we present a flow monitoring extension, called EventFlow, which allows keeping track of relations between HTTP and DNS application flows. Information about flow relation is inserted to flow records to keep track of individual user actions, i.e., events. We develop a prototype of the EventFlow extension and evaluate its properties on network traffic trace from an ISP network. Results show that at least 10 % of HTTP and DNS flow records form complex events. We believe, that this is only a lower bound and that further improvements can be made to relate even more flows into events.

The rest of the section is structured as follows. Related work is surveyed in Subsection 7.1.1. We propose the architecture of EventFlow measurement in Section 7.1.2. Subsection 7.1.3 describes the implementation of the EventFlow prototype. Experimental evaluation of the EventFlow prototype is performed in Subsection 7.1.4. The section is concluded in Subsection 7.1.5.

7.1.1 Related Work

Madhyastha and Krishnamurthy [292] propose a generic language for application-specific flow sampling. Their language allows applications to select flows with special properties so that the negative impact of sampling on these applications is minimised. This can be useful for intrusion detection systems or traffic classification applications. Although the goal of this work is different from ours, it also aims to improve the collected data, so that traffic analysis applications can achieve higher accuracy.

The authors of [293] also focus on improving quality of sampled flow data. They show that the traffic classification accuracy can be increased using related sampling, which assigns a higher probability to connections that are part of the same application. The authors propose to use a source IP address as a measure of the relation between connection sessions.

Hu et al. [294] propose an entropy-based aggregation system to mitigate an impact of DoS attacks and worm spreads on a network monitoring system. The main contribution of their approach is a flow key attribute selection algorithm that chooses key attributes by which the flows are aggregated. A two-dimensional hash table is used to implement their approach. The aggregated flows are called metaflows. The main difference from EventFlow is that we label existing flows belonging to the same user action, while the metaflow is a substitute flow for many flows created during malicious network activity.

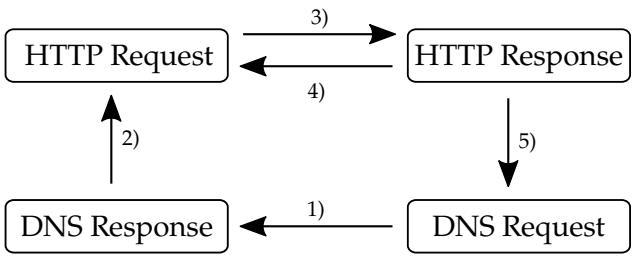


Figure 7.1: Relations between HTTP and DNS Requests and Responses.

Dolberg et al. [295] introduce a multidimensional flow aggregation aimed to reduce the volume of collected data. The authors use tree structures for storing the data by chosen dimension such as IP addresses or ports. EventFlow proposed in our work might be used in this scenario to aggregate flows by the same events.

The usual approach to reducing the volume of collected data is to use sampling. Estan et al. [59] propose to use adaptive sampling rate to achieve highest possible accuracy within given data collection constraints. Their main contribution is a system for renormalisation of flow entries after the sampling rate was changed. The authors propose an extension to standard flow counting that increases the accuracy of the counters for sampled flow.

7.1.2 EventFlow Architecture

This subsection describes the architecture of the EventFlow monitoring. The goal is to label all flows that are the results of a single user action with the same event identifier (EID). For example, accessing `http://www.w3.org/` creates 1 DNS request, 37 HTTP requests, and 8 HTTPS requests. We aim to assign a single unique EID to the flows generated for all of these DNS and HTTP requests.

Four basic types of flows are recognised by the EventFlow: HTTP requests, HTTP responses, DNS requests, and DNS responses. There are relations between these types of flows in network traffic, as shown in Figure 7.1. When HTTP request to a new site is performed, the IP address of the site must be resolved first. Therefore, a DNS request is created. After the request is observed, a reply usually follows, which results in the relation 1). After the DNS reply arrives, the client knows the IP address of the server and makes the HTTP request, which creates the relation 2). An HTTP response follows the request, as indicated by the relation 3). The HTTP response can contain an HTML page which links to several additional resources such as external style sheets or images. The loading of these resources triggers more HTTP requests, resulting in relation 4). When these requests point to previously unresolved domains, new DNS requests are created, which introduces the relation 5).

We base the EventFlow architecture on the relations between the requests and responses. When an HTTP or a DNS flow is encountered, we must make sure that it is assigned the same EID as the related flows. Therefore, we create four sets of records: expected HTTP requests, expected HTTP responses, expected DNS requests, and expected DNS responses. When processing an HTTP or a DNS flow, we add a new record to the set or sets it relates to. Then, when a next flow is processed, it is matched against appropriate expected set to see whether it is a part of existing event. If it is, an EID of the event is assigned to the flow record. For example, when a DNS response is encountered, a new record is put into the expected HTTP requests set (because of relation 2), see Figure 7.1). Then, when an HTTP request is processed, we check the expected HTTP requests set to see whether we are expecting this request based on a previous DNS response. If the request is matched, it is assigned the same EID as the DNS response.

Each of the sets of expected records uses different flow properties to match a flow record. When matching an HTTP request flow against the expected HTTP requests set, the source IP

	Source IP	Destination IP	Destination Port	URL	Domain	DNS Transaction ID
Expected HTTP Request	✓			✓	✓	
Expected HTTP Reply	✓	✓	✓			
Expected DNS Request	✓				✓	
Expected DNS Reply	✓	✓	✓			✓

Table 7.1: Matched Flow Properties.

address of the flow must match as well as the requested domain or the URL, if available. Checking the source IP address ensures that flows from different hosts are not combined into a single event. A domain name is checked for the records that were inserted in the set when DNS reply was encountered. In case an HTTP response caused the record to be inserted, the full URL is available, not only the domain name. Replies are checked based on IP addresses and destination port. Source port is not checked since the services are expected to run on standard, well-known ports. The DNS reply is also checked for transaction ID, which is a unique identifier tying the request and response. However, the DNSSEC extension is ignored and does not affect the EventFlow. Therefore, any malicious responses would still be part of an event. List of the used properties is provided in Table 7.1.

Expiration of the records from the expected sets must be ensured. When a record from any of the expected sets is matched, it is removed. However, many inserted records will never be matched. For example, when a DNS request is made to accommodate a different service than HTTP, the expected HTTP request might never appear. We need to free such records from the sets eventually. A timeout is used to keep the expected sets from being congested by redundant records. A timestamp is assigned to each record upon insertion to a set. Then, records older than the timeout are removed each time the set is searched. The timeout should be as short as possible to avoid blending of several events. However, it should be at least as long as it takes to process the longest user action, which might be up to a couple of seconds in case of complicated queries to slow sites.

There are several caveats to our approach and some limitations of the architecture that should be addressed in the future. Our approach does not handle HTTP redirection codes; therefore the first request and the HTTP 3xx redirection response are assigned different EID than the subsequent request to the resource. This problem can be rectified simply by adding a handler for the HTTP 3xx redirection responses that will put a new record with the redirect URL to the expected HTTP requests set.

Another limitation of our approach is that the URLs are only extracted from HTML documents. However, modern websites often use a JavaScript code to request additional resources through the Ajax technique. Such requests cannot be easily matched to an event since it would require to reconstruct the complete web page and process the included JavaScript code, which is infeasible for the flow monitoring system.

There are also several caveats that cannot be avoided. Some of the requested documents might be cached by clients that would cause EventFlow to lose track of related URLs. However, cached DNS queries are of no consequence to the EventFlow since no traffic is generated for

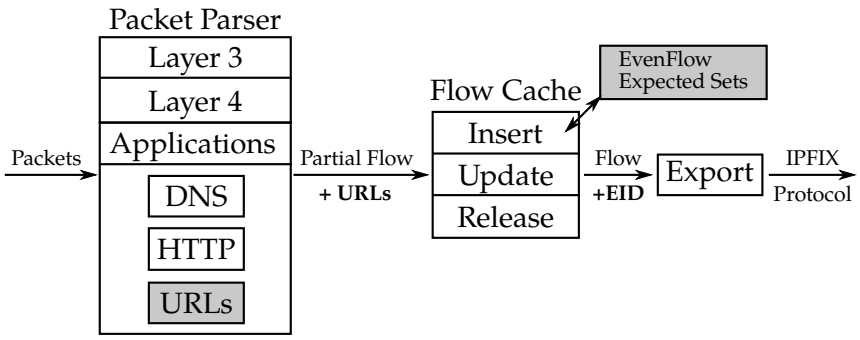


Figure 7.2: EventFlow Prototype Schema.

them and no information about a flow relation is lost. Actions of different users can be mingled when Network Address Translation is used. Finally, the growing deployment of HTTPS reduces the usefulness of the EventFlow for the HTTP protocol. Nevertheless, it can always be used in environments utilising an HTTPS proxy such as data centres or enterprise networks.

7.1.3 EventFlow Prototype

We build the EventFlow prototype as a plugin for the FlowMon [99] flow monitoring software. The capabilities of the flow exporter are utilised to create a prototype EventFlow extension plugin. The prototype can either be deployed to process data on a live network or to analyse captured samples.

The FlowMon exporter consists of three main components as shown in Figure 7.2. The first component is a packet parser. It receives packets from the network and extracts information from a network to application layer of each packet. The extracted information is used to create a partial flow record, which contains all necessary information about the parsed packet such as IP addresses, ports, timestamps, and a byte counter. It also contains application layer information when an application parsing is performed. The partial flow record is passed to the second component of the exporter which is a flow cache. The partial record is either inserted as a new flow record or used to update an existing flow record, which is an aggregation of previous partial flow records. When a flow record expires, it is released from the flow cache to an export component. The purpose of the export component is to convert raw flow records to a flow export protocol format such as NetFlow [21] or IPFIX [37] and pass the flow records over the network for further processing.

Plugins that extend the FlowMon exporter to process specific application layer protocols, such as DNS or HTTP have access to several parts of the flow creation and export process. Each plugin can request to see the raw packet payload, process it, and add its own information to flow records, such as HTTP Host, Content-Type, or Response Code. Furthermore, the plugins are allowed to provide their own functionality for insert, update, and release methods of the flow cache. Lastly, each plugin defines how the information inserted into flow records is processed by the export component.

The EventFlow prototype is implemented as an application protocol extension. However, it also utilises the data provided by other application plugins. The EventFlow combines information from the DNS and HTTP protocols to detect relations between flows; therefore it requires the DNS and HTTP application plugins to be deployed as well. The prototype extends the packet parser to extract URLs from HTML pages. These URLs are sent together with partial flow record to the flow cache. However, they are used only internally, and they are newer exported in the flow records. When a new flow record is created in the flow cache, the expected sets (see Section 7.1.2) are searched for a match to the new record using DNS, HTTP, and URL information

provided in the partial record. If a match is found, the new flow records are assigned an Event ID (8-byte unsigned integer) of the matched record from the expected sets. Otherwise, if no match is found, a new EID is incrementally assigned to the new flow record. After the flow is expired from the cache, the EID is a part of the record and is sent by the export component along with the rest of the flow record.

Using an 8-byte integer for EID and assigning it incrementally to individual events ensures that there are no collisions due to EID overflow in practice. However, an assignment that is individual for each flow probe and persistent over the reboots of the system would be required for a real-world deployment. The EID is assigned only to flows of the HTTP and DNS traffic since it would provide no benefit to other flows as event relation tracking is not implemented for other protocols yet. Moreover, the size of the flows grows only by 8 bytes at maximum, which has a negligible impact on the flow collector disk space requirements.

7.1.4 Experimental Evaluation

We evaluate the prototype in two scenarios. First, we assess the functionality of the prototype on a simple example web site. Once we have verified the functionality, we run EventFlow on a packet trace from live network to determine how many flows can be joined to events in real traffic. The IPFIXcol [A12] flow collector is used to collect and process the generated flows. The main advantage of the collector is that it can be easily configured to work with the new Event ID element.

We do not evaluate the performance of the prototype in this phase. We are aware of several performance inefficiencies that need to be solved before any valuable results can be measured. For example, one of the most expensive parts of the prototype is the management of sets of expected records. We expect that changing the underlying data structures will significantly improve the performance.

Functional Evaluation

For the first scenario, we create a simple website with two pages, each linking the other page, displaying an image, and referencing a different JavaScript library. The evaluation proceeds as follows. We request the first page in a browser and few seconds after it loads we follow the link to the other page. The packet trace of these actions is recorded and processed by the EventFlow prototype, and the resulting flows are collected by the IPFIXcol.

We expect to see a flow record for each of the requests and responses. However, due to the HTTP pipelining the whole communication with the web server hosting the test pages is done using a single connection. Therefore, there is a pair of flows for the accessing the two web pages with the linked images (which were on the same server), two pairs of flows for each off-site JavaScript library, and three pairs of DNS flows for IP address resolution. There are 12 flows created in this test scenario in total. The 12 flows are divided into two events by the EventFlow prototype. The first event contains flows for the two DNS requests, HTTP communication with the web server and download of the first JavaScript library. The second event does not contain an HTTP flow due to the HTTP pipelining but contains the DNS request and the subsequent download of the second JavaScript library.

The functional evaluation shows that the prototype correctly recognises related flows and labels them as a part of the same event. The flow exporter can be extended to handle HTTP pipelining by creating new flow record for each pipelined request. Such extension would make the measurement more accurate.

Total Flows	613,953
HTTP Requests	33,294
HTTP Responses	49,753
DNS Requests	197,926
DNS Responses	224,588
Events with > 1 Flow	28,064
Flows in Events with > 1 Flow	55,881
All Events	388,749
Flows in All Events	418,671

Table 7.2: Real Traffic Evaluation Statistics.

Real Traffic Evaluation

The purpose of the real traffic test is to determine how many flows can be joined into events. We collect a short (approximately one minute) trace of 10 million packets from an ISP network on ports 53 and 80 which are likely to be DNS and HTTP packets. Table 7.2 shows statistics that describe the packet trace as well as the results of the evaluation. We can see that from the total number of more than 600 thousand flows more than 400 thousand are part of events. Furthermore, over 55 thousand flows are part of events which contain more than one flow. Therefore, we can conclude that more than 10 % of observed HTTP and DNS flows are recognised as a part of more complex events by the EventFlow prototype.

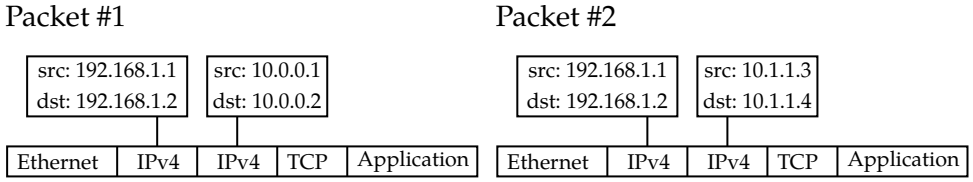
The number of flows in complex events is not as high as might be expected given the large number of HTTP and DNS requests and responses. We believe that this is caused by a quite short time window of our trace, which is likely to have captured a large number of separate responses and requests. Moreover, we believe that better results can be achieved by fine-tuning the timeout of the records in the expected sets of the EventFlow prototype.

7.1.5 Conclusions

We have presented an EventFlow monitoring architecture that allows keeping track of relations between HTTP and DNS application flows, which can be used to simplify behavioural analysis of network traffic, improve network threat detection and network traffic classification. The changes to existing flow monitoring architecture are negligible, which facilitates wide deployment. The relation between flows is encoded as an 8-byte unsigned integer called Event ID which is shared by the related flows. The proposed architecture can be further extended to handle more complex HTTP communication, such as redirection return codes.

A prototype of EventFlow plugin for the FlowMon flow exporter has been evaluated on a trace of 10 million packets. We showed that more than 10 % of observed HTTP and DNS flows are recognised as a part of more complex events by our prototype. We believe that this result will improve on longer packet trace as well as with more accurate settings of the prototype. Prospective improvements to the prototype as well as its more detailed evaluation, including a performance evaluation, are left for a future work.

We believe that the network analysis will benefit from the supplemental information about flow relations. Our work has shown that it is possible to acquire such information without a significant impact on an existing monitoring architecture and that it is possible to extend the flow monitoring to trace relations of other application protocols.



Application flow records

src IPv4	dst IPv4	src IPv4	dst IPv4	packets
192.168.1.1	192.168.1.2	10.0.0.1	10.0.0.2	1
192.168.1.1	192.168.1.2	10.1.1.3	10.1.1.4	1

Tree structure of flow records

src IPv4	dst IPv4	packets	ID	parent ID
192.168.1.1	192.168.1.2	2	#1	-
10.0.0.1	10.0.0.2	1	#2	#1
10.1.1.3	10.1.1.4	1	#3	#1

Figure 7.3: MetaFlow Tree Structure.

7.2 MetaFlow

The MetaFlow concept aims to provide support for capturing relations between flows observed in encapsulated traffic, such as tunnels. The widely used approach for monitoring encapsulated traffic is to combine flow keys of different layers to create a new flow key. This flow key splits the flow containing the encapsulated traffic into multiple flows based on the encapsulated traffic. An example of this approach is shown in Figure 7.3.

We propose to use a more generic approach to handle the encapsulated traffic. In our approach, when an encapsulated traffic is encountered, each layer is treated as a separate flow. Therefore, when a tunnel contains two different communications, we create three different flows. To ensure that the relation between the encapsulation layers is preserved, each flow contains *ID* and *Parent ID* elements, as shown in Figure 7.3. Therefore, the encapsulated traffic is represented by a tree structure.

The main benefit of the MetaFlow approach is that the structure of flows remains the same for an arbitrary number of encapsulation layers. Current flow exporters usually store only the innermost and outermost layers while ignoring any intermediate layers altogether. By providing a uniform way to describe more complex packet structures, we can gain more information about the traffic. Moreover, the flow records are easier to process as well. For example, it is simple to select all encapsulated flows by selecting all flows with non-empty *Parent ID*.

The uniform description allows running queries and algorithms on encapsulated traffic with no additional changes in the flow data processing. When the encapsulated traffic is monitored nowadays, both sets of IP addresses that are present in flow records must be handled, which creates complicated conditions in the code. However, by simplifying the flow records, this problem is removed. Another advantage of the simplified flow records is the lowered number of templates created during flow export. Currently, each layer of encapsulation creates a whole new set of templates that match the templates generated without encapsulation. For example, IPv4 packets with ICMP, UDP, and TCP transport layer protocols usually create three different templates. When the IPv4 header encapsulates another IPv4 protocol header, three new templates are created for ICMP, UDP, and TCP protocols observed in the encapsulated traffic. Handling of a large number of templates is inefficient both for flow export and flow data processing. The MetaFlow removes this inefficiency as no new templates are added at all in this scenario.

To implement MetaFlow, changes would need to be made both to flow data creation and flow data processing. The main modification is that each flow needs to have a unique *ID*. There are several options for generating a unique identifier for each flow during the flow creation process:

- A sequence number,
- a high-precision timestamp,
- a hash.

Using a sequence number is fine as long as only a single flow cache is utilised. Shared sequence number means unnecessary synchronisation point for multiple flow caches, which could decrease performance. Moreover, when the exporter is restarted, the numbers would have to continue in the sequence, which is hard to implement in case of sudden reboots and crashes. If the sequence was restarted, the flow collector would have to detect it and renumber the flows, which would lead to consistency problems.

Using high-precision timestamps is fine as long as each consecutive packets are guaranteed to receive different timestamp. If this condition is met, each flow can simply get its start timestamp as its *ID*. For example, the precision needed for 100 Gbps network would need to be in nanoseconds.

Another option is to utilise the same hash that is used to identify the flow during the flow creation process. However, the hash needs to change when the flow is exported and never be used again. Therefore, the best option seems to be using high-precision timestamps. When they are not available, a combination of hash mixed together with a timestamp should suffice.

Once a flow is received by a collector, the *ID* and *Parent ID* needs to be used together with an identification of the flow exporter. Otherwise, the flows might be mixed as there is no guarantee that each exported generates unique identifiers.

The implementation of flow data processing would need to be reviewed to ensure compatibility with the MetaFlow concept. However, we expect that only minimal changes to flow processing algorithms would need to be done to handle the new format.

7.3 Application Events and Flow

A concept similar to MetaFlow can be applied to application events as well. Currently, an application flow can be split into multiple flows based on the application content, e.g., multiple HTTP requests on a single connection in separate flows. The result is the same as for encapsulated traffic. Statistical properties of basic flows are broken by the application layer in these cases. Moreover, mixing together basic flow which focuses on network and transport layers and the application data forces too narrow view of the available data. We propose to separate application layer data from the basic flow records in the same fashion as MetaFlow separates encapsulated layers from the parent flow. This would effectively decouple application events from the basic flows.

There are three major benefits to our proposal. Firstly, the basic flows remain untouched whatever the application layer information is. This allows detection and statistical methods to rely on the properties of the basic flows without taking into account possible transformation by the application layer. Moreover, when the application layer is not necessary for flow processing, it can easily be filtered out without modification of each flow record. This might lead to major performance benefits for the flow data processing systems.

Secondly, the flow exporter does not have to hold the application information in the flow cache. The flow cache can be implemented efficiently with small flow records without variable fields and dynamically allocated memory. The processing of application events would be moved to packet processing. If the event has no duration, it can be exported directly without delay. We

assume that the associated flow identifier is known from the time the flow records is created in the flow cache and therefore can be assigned to the application event at any time. If the application event has a duration, the packet parser must store it in an appropriate application-specific cache until the time it is completed and exported. We expect this mode of data processing at the flow exporter to be significantly more effective than the current mode which holds all application data in flow cache together with the flow records.

The last major advantage is in the processing of the application events. Since each application event is exported separately, the number of templates is significantly reduced. Instead of many combinations of network, transport, and application layers, we only have combinations of network and transport layers and then separate templates for application events. Moreover, this separation can simplify the processing of the flows as well. For example, when a detection module requires only data from HTTP and DNS protocols, it can easily select only the two templates to work with.

The proposed approach has one notable disadvantage. The flow processing systems are going to receive application events before the flow records expire. This might cause them to buffer the data and wait for the flow records. However, basic information about each flow, such as IP addresses, transport protocol, and port numbers can be included in the application event messages as well. This should facilitate unhindered processing of the flow events while the flow records are being aggregated. Moreover, detection methods using application events could react much faster than those utilising application flow records.

Since the application data are effectively decoupled from the basic flows, we can quite easily inject data from another source, such as application logs to the data stream. The injected data can share the same template with the application events, which would facilitate the shared processing. Moreover, based on timestamps and network information, the injected data could be attached to the basic flows as well. There are several reasons to do this. Firstly, the logs can provide ground truth for the application layer data. Secondly, having a single system processing all available data from the network and host systems helps to improve quality of data and accuracy of various detection algorithms. Lastly, using a single system to process data from multiple sources reduces management and running costs.

7.4 Summary

We have proposed three novel concepts for application flow monitoring in this chapter. Each of our proposals could be implemented separately; however, we believe that a combination of all proposed approaches is not only possible but would provide greatest benefits for the application flow monitoring.

The first proposed concept has been EventFlow. It is an extension of the application flow monitoring which allows preserving relations between HTTP and DNS application flows that are a part of single user action, most typically browsing a web page. We have described an architecture of the EventFlow extension and its limitations. A prototype implementation of the EventFlow has been introduced and evaluated on a packet trace from an ISP network. We have shown that a significant number of flow records can be recognised as a part of a single user action.

MetaFlow has been proposed as an approach to monitoring of encapsulated traffic. We have argued that the current practice of extending flow records with information from each layer is not flexible and has several disadvantages. MetaFlow aims to create a hierarchy of encapsulated flows so that each flow is able to retain a simple structure while the information about the relations between flows is preserved. We have also discussed how to create a unique flow identifier, which is necessary for building the MetaFlow tree structure.

The last proposal has expanded the MetaFlow concept to application layer events. By decoupling application events from basic flow records, we can significantly simplify both flow creation process and flow data processing systems. Moreover, it allows us to solve issues produced by the forceful splitting of flows based on application payload. We believe that separating application events from basic flow monitoring is the future of application flow monitoring. Moreover, this approach provides new opportunities for novel research not only for flow monitoring but also for flow data processing systems.

Chapter 8

Conclusion

The flow monitoring has evolved significantly in the past several years. The most substantial progress has been made in the performance area of flow monitoring, which needed to match the development of networking toward 100 Gbps technologies, and in the processing of application layer information. This thesis has presented our contributions in both of these areas.

8.1 Research Goals

We have established the following three research goals in the introduction of this thesis:

- Propose application flow monitoring which utilises application layer information to facilitate flow analysis and threat detection.
- Evaluate performance of flow monitoring and propose optimisations to facilitate monitoring of high-speed networks.
- Analyse options for monitoring of encrypted traffic, survey common encryption protocols and methods for encrypted traffic classification.

To address the first goal, we have proposed and defined application-aware flow monitoring (called simply application flow monitoring). We started by analysing the current definition of flow, provided an improved alternative, and formalised our definition. Based on the definition of flow, we have proposed a definition of application flow. Since the use of the terms flow, IP flow, and application flow differs in the literature, we have introduced a consistent terminology to deal with the potentially confusing differences. After that, we have described the application-aware flow monitoring system and pointed out the important differences from the basic flow.

To show the value of application flow monitoring for traffic analysis, we have analysed four different use cases. The first, most common use case, has utilised extended information from HTTP headers to detect new classes of attack on the application layer. The second use case has shown how IPv6 tunnelled traffic can be observed and analysed with application flow monitoring. We have demonstrated that the newly acquired information helps us to understand the network traffic better. The third use case has added geolocation information to flow records to aid traffic analysis, and the fourth discussed how different samples of network traffic might be compared and experiments repeated on a live network.

Our work on the second research goal has included both application and basic flow monitoring. We have designed and implemented application flow monitoring for HTTP protocol and evaluated its performance. Multiple approaches to the design of HTTP application parser were taken into consideration and utilised. By comparing results of different parser implementations, we have shown that the performance of application flow monitoring system is significantly affected by the design of application parsers. Moreover, the composition of the traffic matters even

more than for the basic flow as the percentage of the given application in the traffic affects the performance substantially.

We have discussed how the design of the flow monitoring process affects the performance. Performing application analysis for multiple protocols at speeds of and beyond 100 Gbps is not possible without heavy acceleration using specialised NICs [223], and even then depends on the traffic composition. We have build a high-density flow monitoring system, which could process traffic from sixteen 10 Gbps links, and analysed its performance. We compared settings with and without hardware acceleration for different packet sizes and different numbers of simultaneous flows. Our results have shown that full line rate can be achieved on real networks. We believe that the results could be improved by applying more of the described optimisations.

The last of our research goals has aimed to address the growing use of encryption, which prevents analysis of application payloads. We had created a comprehensive overview of methods for classification and analysis of encrypted traffic. Firstly, we have described the most widely used encryption protocols and shown that each communication starts with an unencrypted phase which allows us to gather important data. Moreover, the extensive use of X.509 certificates contributes to the amount of information that can be obtained from the unencrypted phase. This information can be included in flow records and used for analysis of encrypted traffic. The second part of our work has surveyed methods for classification and analysis of encrypted traffic. We have discovered that most of the methods described in the literature use supervised or semi-supervised machine learning algorithms to classify flows and even determine the exact application protocol for each flow. Most methods target SSH, SSH/TLS, and BitTorrent encryption protocols. Although we have categorised the described classification methods, we did not attempt to apply them directly to flow monitoring process. Evaluation of the classification methods for use with flow monitoring remains as future work.

8.2 Further Research

Even though this thesis significantly contributed to the state-of-the-art of the flow monitoring, the changing nature of network traffic continually opens new research possibilities. We list the following topics as interesting for future research:

- Flow monitoring at speeds of 400 Gbps and beyond. The commodity hardware is likely not going to be able to process this amount of data in the near future without a significant change of packet capture paradigm or use of hardware acceleration. Building a flow monitoring system for the rates of hundreds of gigabits per second is definitely a challenging topic.
- The use of encrypted traffic classification as a part of flow monitoring is an increasingly interesting topic due to the continuous rise in the encrypted traffic volume. The main challenge is to find an accurate method that is fast enough to be used at high speeds on live networks.
- An increasing amount of applications are being hosted using cloud services. However, the customer cannot easily deploy flow monitoring in the cloud to receive flow data. Therefore, the challenge is to propose flow monitoring solution that enables users to receive necessary flow data, possibly as a part of the service. Host-based flow monitoring extended by information from the system and application logs also seems to be a viable and interesting option.
- Flow monitoring in a virtual environment is an active research topic nowadays. Although most proposals to use OpenFlow protocol are not suitable for high-speed networks and do not produce quality flow data, it is an opportunity for future research.

- With the increasing amount of processed data, the number of flow records and their lengths increase as well. The flow data processing systems require more and more performance and use of distributed architecture to be able to cope with the data. We propose to evaluate how the flow data are used by the flow data processing systems. Then, create a loopback to the flow monitoring systems and eliminate the creation and processing of the data, that is dispensable. By studying the interaction between the flow monitoring and flow data processing systems we can optimise both processes to be not only cheaper but also more accurate.

The concepts introduced in Chapter 7 should help with several of the proposed research topics. Especially the separation of application events and flow records should prove useful for host-based monitoring of virtual hosts in a cloud and analysis of requirements of the flow data processing systems.

Bibliography

Authored publications referenced in the thesis

[A1] Petr Velan. "Improving Network Flow Definition: Formalization and Applicability". In: *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*. 2018, Accepted for publication, 5 pages (page 5).

[A2] Petr Velan. "Practical Experience with IPFIX Flow Collectors". In: *IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*. Ghent, Belgium: IEEE Xplore Digital Library, May 2013, pp. 1021–1026. ISBN: 978-1-4673-5229-1 (page 29).

[A3] Petr Velan, Milan Čermák, Pavel Čeleda, and Martin Drašar. "A Survey of Methods for Encrypted Traffic Classification and Analysis". In: *International Journal of Network Management* 25.5 (2015), pp. 355–374. DOI: 10.1002/nem.1901 (pages 30, 127).

[A4] Petr Velan, Tomáš Jirsík, and Pavel Čeleda. "Design and Evaluation of HTTP Protocol Parsers for IPFIX Measurement". In: *Advances in Communication Networking*. Vol. 8115. Heidelberg: Springer Berlin Heidelberg, 2013, pp. 136–147. ISBN: 978-3-642-40551-8 (pages 35, 57).

[A5] Petr Velan and Pavel Čeleda. "Next Generation Application-Aware Flow Monitoring". English. In: *Monitoring and Securing Virtualized Networks and Services*. Vol. 8508. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2014, pp. 173–178. ISBN: 978-3-662-43861-9 (page 35).

[A6] Martin Husák, Petr Velan, and Jan Vykopal. "Security Monitoring of HTTP Traffic Using Extended Flows". In: *2015 10th International Conference on Availability, Reliability and Security*. August 2015, pp. 258–265 (page 53).

[A7] Martin Elich, Petr Velan, Tomáš Jirsík, and Pavel Čeleda. "An Investigation Into Teredo and 6to4 Transition Mechanisms: Traffic Analysis". In: *IEEE 38th Conference on Local Computer Networks Workshops (LCN Workshops)*. Sydney, Australia: IEEE Xplore Digital Library, October 2013, pp. 1046–1052. ISBN: 978-1-4799-0540-9 (page 53).

[A8] Pavel Čeleda, Petr Velan, Martin Rábek, Rick Hofstede, and Aiko Pras. "Large-Scale Geolocation for NetFlow". In: *IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*. Ghent, Belgium: IEEE Xplore Digital Library, May 2013, pp. 1015–1020. ISBN: 978-1-4673-5229-1 (page 53).

[A9] Petr Velan, Jana Medková, Tomáš Jirsík, and P. Čeleda. "Network Traffic Characterisation Using Flow-Based Statistics". In: *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*. April 2016, pp. 907–912 (page 53).

[A10] Luuk Hendriks, Petr Velan, Ricardo de Oliveira Schmidt, Pieter-Tjerk de Boer, and Aiko Pras. "Flow-Based Detection of IPv6-specific Network Layer Attacks". In: *Security of Networks and Services in an All-Connected World: 11th IFIP WG 6.6 International Conference on Autonomous Infrastructure, Management, and Security, AIMS 2017, Zurich, Switzerland, July 10-13, 2017, Proceedings*. Cham: Springer International Publishing, 2017, pp. 137–142. ISBN: 978-3-319-60774-0 (page 53).

[A11] Luuk Hendriks, Petr Velan, Ricardo de Oliveira Schmidt, Pieter-Tjerk de Boer, and Aiko Pras. "Threats and Surprises behind IPv6 Extension Headers". In: *2017 Network Traffic Measurement and Analysis Conference (TMA)*. IEEE Xplore Digital Library, June 2017, pp. 1–9 (page 53).

[A12] Petr Velan and Radek Krejčí. "Flow Information Storage Assessment Using IPFIXcol". In: *Dependable Networks and Services*. Vol. 7279. Lecture Notes in Computer Science. Heidelberg: Springer Berlin Heidelberg, 2012, pp. 155–158. ISBN: 978-3-642-30632-7 (pages 57, 70, 152).

[A13] Petr Velan and Viktor Puš. "High-Density Network Flow Monitoring". In: *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. May 2015, pp. 996–1001 (page 99).

[A14] Viktor Puš, Petr Velan, Lukáš Kekely, Jan Kořenek, and Pavel Minařík. "Hardware Accelerated Flow Measurement of 100 Gb Ethernet". eng. In: *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. Ottawa, Canada: IEEE Xplore Digital Library, May 2015, pp. 1147–1148 (page 99).

[A15] Petr Velan. "EventFlow: Network Flow Aggregation Based on User Actions". In: *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*. April 2016, pp. 767–771 (page 147).

Other publications referenced in the thesis

[1] Cisco Systems, Inc., San Jose, CA and USA. *Cisco IOS NetFlow and Security*. February 2005. URL: http://www.cisco.com/en/US/prod/collateral/iosswrel/ps6537/ps6586/ps6642/prod_presentation0900aecd80311f49.pdf (Accessed on 27th April 2017) (pages 1, 7).

[2] Ward van Wanrooij and Aiko Pras. "Data on Retention". In: *Proceedings of the 16th IFIP/IEEE Ambient Networks International Conference on Distributed Systems: Operations and Management*. DSOM'05. Barcelona, Spain: Springer-Verlag, 2005, pp. 60–71. ISBN: 978-3-540-29388-0 (page 1).

[3] Center for Strategic and International Studies. *The Economic Impact of Cubercrime and Cyber Espionage*. July 2013. URL: http://csis.org/files/publication/60396rpt_cybercrime-cost_0713_ph4_0.pdf (Accessed on 2nd March 2018) (page 1).

[4] Cisco Systems, Inc., San Jose, CA and USA. *Cisco IOS Flexible NetFlow*. December 2008. URL: http://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/flexible-netflow/product_data_sheet0900aecd804b590b.html (Accessed on 27th April 2017) (pages 1, 7).

[5] Cisco Systems, Inc., San Jose, CA and USA. *Network Based Application Recognition (NBAR)*. URL: https://www.cisco.com/c/en/us/products/ios-nx-os-software/network-based-application-recognition-nbar/index.html (Accessed on 2nd March 2018) (pages 1, 136).

[6] Jessica Steinberger, Lisa Schehlmann, Sebastian Abt and Harald Baier. "Anomaly Detection and Mitigation at Internet Scale: A Survey". In: *Proceedings of the 7th IFIP WG 6.6 International Conference on Autonomous Infrastructure, Management, and Security: Emerging Management Mechanisms for the Future Internet - Volume 7943*. AIMS'13. Barcelona, Spain: Springer-Verlag, 2013, pp. 49–60. ISBN: 978-3-642-38997-9 (page 1).

[7] Daniela Brauckhoff, Bernhard Tellenbach, Arno Wagner, Martin May and Anukool Lakhina. "Impact of Packet Sampling on Anomaly Detection Metrics". In: *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*. IMC '06. Rio de Janeriro, Brazil: ACM, 2006, pp. 159–164. ISBN: 1-59593-561-4 (pages 1, 107).

[8] Rick Hofstede, Idilio Drago, Anna Sperotto, Ramin Sadre and Aiko Pras. "Measurement Artifacts in NetFlow Data". In: *Passive and Active Measurement*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 1–10. ISBN: 978-3-642-36516-4 (page 1).

[9] Cisco Systems, Inc., San Jose, CA and USA. *Cisco Visual Networking Index: Forecast and Methodology, 2016–2021*. June 2017. URL: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.pdf (Accessed on 2nd March 2018) (page 1).

[10] ntop. *Unveiling Application Visibility in ntop and nProbe (both in NetFlow v9 and IPFIX)*. November 2011. URL: http://www.ntop.org/nprobe/unveiling-application-visibility-in-ntop-and-nprobe-both-in-netflow-v9-and-ipfix/ (Accessed on 27th September 2017) (pages 2, 37).

[11] Yves Younan. *25 Years of Vulnerabilities: 1988-2012*. 2013. URL: https://courses.cs.washington.edu/courses/cse484/14au/reading/25-years-vulnerabilities.pdf (Accessed on 2nd March 2018) (page 2).

[12] Luca Deri. "Passively Monitoring Networks at Gigabit Speeds Using Commodity Hardware and Open Source Software". In: *Proceedings of the Passive and Active Measurement Conference*. 2003, pp. 1–7 (page 2).

[13] Nevena Vratonjic, Julien Freudiger, Vincent Bindschaedler and Jean-Pierre Hubaux. "The Inconvenient Truth About Web Certificates". In: *Economics of Information Security and Privacy III*. New York, NY: Springer New York, 2013, pp. 79–117. ISBN: 978-1-4614-1981-5 (page 2).

[14] Internet Security Research Group (ISRG). *Let's Encrypt Stats*. March 2018. URL: https://letsencrypt.org/stats/ (Accessed on 2nd March 2018) (page 2).

[15] Rick Hofstede, Pavel Čeleda, Brian Trammell, Idilio Drago, Ramin Sadre, Anna Sperotto and Aiko Pras. "Flow Monitoring Explained: From Packet Capture to Data Analysis with NetFlow and IPFIX". In: *Communications Surveys Tutorials, IEEE* PP.99 (2014), pp. 2037–2064. ISSN: 1553-877X. DOI: 10.1109/COMST.2014.2321898 (pages 5, 16, 26, 37, 56, 86, 89, 101).

[16] C. Mills, D. Hirsh and G.R. Ruth. *Internet Accounting: Background*. RFC 1272 (Informational). RFC. Fremont, CA, USA: RFC Editor, November 1991. URL: https://www.rfc-editor.org/rfc/rfc1272.txt (page 6).

[17] S. Farrell and H. Tschofenig. *Pervasive Monitoring Is an Attack*. RFC 7258 (Best Current Practice). RFC. Fremont, CA, USA: RFC Editor, May 2014. URL: https://www.rfc-editor.org/rfc/rfc7258.txt (pages 6, 146).

[18] Kimberly C. Claffy, Hans-Werner Braun and George C. Polyzos. "A Parameterizable Methodology for Internet Traffic Flow Profiling". In: *IEEE Journal on Selected Areas in Communications* 13.8 (October 1995), pp. 1481–1494. ISSN: 0733-8716. DOI: 10.1109/49.464717 (page 6).

[19] N. Brownlee, C. Mills and G. Ruth. *Traffic Flow Measurement: Architecture*. RFC 2722 (Informational). RFC. Fremont, CA, USA: RFC Editor, October 1999. URL: https://www.rfc-editor.org/rfc/rfc2722.txt (page 6).

[20] Cisco Systems, Inc., San Jose, CA and USA. *NetFlow Services Solutions Guide*. January 2007. URL: http://www.cisco.com/en/US/docs/ios/solutions_docs/netflow/nfwhite.html (Accessed on 27th April 2017) (pages 6–8, 86).

[21] B. Claise. *Cisco Systems NetFlow Services Export Version 9*. RFC 3954 (Informational). RFC. Fremont, CA, USA: RFC Editor, October 2004. URL: https://www.rfc-editor.org/rfc/rfc3954.txt (pages 7, 10, 151).

[22] The Internet Engineering Steering Group. *IP Flow Information Export (ipfix) Charter*. URL: http://datatracker.ietf.org/wg/ipfix/charter/ (Accessed on 27th April 2017) (page 7).

[23] The Internet Engineering Steering Group. *IP Flow Information Export Charter*. September 2001. URL: https://www.ietf.org/mail-archive/web/ipfix/current/msg00213.html (Accessed on 27th April 2017) (page 7).

[24] J. Quittek, T. Zseby, B. Claise and S. Zander. *Requirements for IP Flow Information Export (IPFIX)*. RFC 3917 (Informational). RFC. Fremont, CA, USA: RFC Editor, October 2004. URL: https://www.rfc-editor.org/rfc/rfc3917.txt (pages 7, 24).

[25] S. Leinen. *Evaluation of Candidate Protocols for IP Flow Information Export (IPFIX)*. RFC 3955 (Informational). RFC. Fremont, CA, USA: RFC Editor, October 2004. URL: https://www.rfc-editor.org/rfc/rfc3955.txt (page 8).

[26] Brian Trammell and Elisa Boschi. "An Introduction to IP Flow Information Export (IPFIX)". In: *IEEE Communications Magazine* 49.4 (April 2011), pp. 89–95. ISSN: 0163-6804. DOI: 10.1109/MCOM.2011.5741152 (page 8).

[27] B. Trammell and E. Boschi. *Bidirectional Flow Export Using IP Flow Information Export (IPFIX)*. RFC 5103 (Proposed Standard). RFC. Fremont, CA, USA: RFC Editor, January 2008. URL: https://www.rfc-editor.org/rfc/rfc5103.txt (pages 8, 24).

[28] G. Sadasivan, N. Brownlee, B. Claise and J. Quittek. *Architecture for IP Flow Information Export*. RFC 5470 (Informational). RFC. Updated by RFC 6183. Fremont, CA, USA: RFC Editor, March 2009. URL: https://www.rfc-editor.org/rfc/rfc5470.txt (pages 8, 15–17, 20, 22, 23).

[29] E. Boschi, L. Mark and B. Claise. *Reducing Redundancy in IP Flow Information Export (IPFIX) and Packet Sampling (PSAMP) Reports*. RFC 5473 (Informational). RFC. Fremont, CA, USA: RFC Editor, March 2009. URL: https://www.rfc-editor.org/rfc/rfc5473.txt (page 8).

[30] T. Dietz, A. Kobayashi, B. Claise and G. Muenz. *Definitions of Managed Objects for IP Flow Information Export*. RFC 5815 (Proposed Standard). RFC. Obsoleted by RFC 6615. Fremont, CA, USA: RFC Editor, April 2010. URL: https://www.rfc-editor.org/rfc/rfc5815.txt (page 8).

[31] T. Dietz, A. Kobayashi, B. Claise and G. Muenz. *Definitions of Managed Objects for IP Flow Information Export*. RFC 6615 (Proposed Standard). RFC. Fremont, CA, USA: RFC Editor, June 2012. URL: https://www.rfc-editor.org/rfc/rfc6615.txt (page 8).

[32] P. Aitken, B. Claise, S. B S, C. McDowall and J. Schoenwaelder. *Exporting MIB Variables Using the IP Flow Information Export (IPFIX) Protocol*. RFC 8038 (Proposed Standard). RFC. Fremont, CA, USA: RFC Editor, May 2017. URL: https://www.rfc-editor.org/rfc/rfc8038.txt (page 8).

[33] A. Kobayashi and B. Claise. *IP Flow Information Export (IPFIX) Mediation: Problem Statement*. RFC 5982 (Informational). RFC. Fremont, CA, USA: RFC Editor, August 2010. URL: https://www.rfc-editor.org/rfc/rfc5982.txt (page 8).

[34] A. Kobayashi, B. Claise, G. Muenz and K. Ishibashi. *IP Flow Information Export (IPFIX) Mediation: Framework*. RFC 6183 (Informational). RFC. Fremont, CA, USA: RFC Editor, April 2011. URL: https://www.rfc-editor.org/rfc/rfc6183.txt (pages 8, 14, 15).

[35] E. Boschi and B. Trammell. *IP Flow Anonymization Support*. RFC 6235 (Experimental). RFC. Fremont, CA, USA: RFC Editor, May 2011. URL: https://www.rfc-editor.org/rfc/rfc6235.txt (page 8).

[36] G. Muenz, B. Claise and P. Aitken. *Configuration Data Model for the IP Flow Information Export (IPFIX) and Packet Sampling (PSAMP) Protocols*. RFC 6728 (Proposed Standard). RFC. Fremont, CA, USA: RFC Editor, October 2012. URL: https://www.rfc-editor.org/rfc/rfc6728.txt (page 8).

[37] B. Claise, B. Trammell and P. Aitken. *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information*. RFC 7011 (Internet Standard). RFC. Fremont, CA, USA: RFC Editor, September 2013. URL: https://www.rfc-editor.org/rfc/rfc7011.txt (pages 9, 10, 15, 16, 26, 28, 34, 57, 151).

[38] Nevil Brownlee. "Flow-Based Measurement: IPFIX Development and Deployment". In: *IEICE Transactions on Communications* E94.B.8 (September 2011), pp. 2190–2198. DOI: 10.1587/transcom.E94.B.2190 (pages 9, 16).

[39] Peter Phaal. *sFlow Version 5*. 2004. URL: http://sflow.org/sflow_version_5.txt (Accessed on 27th April 2017) (page 9).

[40] P. Phaal, S. Panchen and N. McKee. *InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks*. RFC 3176 (Informational). RFC. Fremont, CA, USA: RFC Editor, September 2001. URL: https://www.rfc-editor.org/rfc/rfc3176.txt (page 9).

[41] Internet Engineering Task Force. *Packet Sampling (psamp) WG*. URL: https://datatracker.ietf.org/wg/psamp/ (Accessed on 27th April 2017) (page 9).

[42] The Internet Engineering Steering Group. *Packet Sampling Charter*. URL: https://datatracker.ietf.org/doc/charter-ietf-psamp/ (Accessed on 27th April 2017) (page 9).

[43] T. Dietz, B. Claise, P. Aitken, F. Dressler and G. Carle. *Information Model for Packet Sampling Exports*. RFC 5477 (Proposed Standard). RFC. Fremont, CA, USA: RFC Editor, March 2009. URL: https://www.rfc-editor.org/rfc/rfc5477.txt (page 9).

[44] T. Zseby, M. Molina, N. Duffield, S. Niccolini and F. Raspall. *Sampling and Filtering Techniques for IP Packet Selection*. RFC 5475 (Proposed Standard). RFC. Fremont, CA, USA: RFC Editor, March 2009. URL: https://www.rfc-editor.org/rfc/rfc5475.txt (page 9).

[45] B. Claise, A. Johnson and J. Quittek. *Packet Sampling (PSAMP) Protocol Specifications*. RFC 5476 (Proposed Standard). RFC. Fremont, CA, USA: RFC Editor, March 2009. URL: https://www.rfc-editor.org/rfc/rfc5476.txt (pages 9, 22).

[46] Open Networking Foundation. *OpenFlow Switch Specification*. September 2012. URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.1.pdf (Accessed on 27th April 2017) (page 9).

[47] Sanjeev Singh and Rakesh Kumar Jha. "A Survey on Software Defined Networking: Architecture for Next Generation Network". In: *J. Netw. Syst. Manage.* 25.2 (April 2017), pp. 321–374. ISSN: 1064-7570. DOI: 10.1007/s10922-016-9393-9 (page 9).

[48] Fei Hu, Qi Hao and Ke Bao. "A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation". In: *IEEE Communications Surveys Tutorials* 16.4 (2014), pp. 2181–2206. ISSN: 1553-877X. DOI: 10.1109/COMST.2014.2326417 (page 9).

[49] Curtis Yu, Cristian Lumezanu, Yueping Zhang, Vishal Singh, Guofei Jiang and Harsha V. Madhyastha. "FlowSense: Monitoring Network Utilization with Zero Measurement Cost". In: *Proceedings of the 14th International Conference on Passive and Active Measurement*. PAM'13. Hong Kong, China: Springer-Verlag, 2013, pp. 31–41. ISBN: 978-3-642-36515-7 (page 9).

[50] Luuk Hendriks, Ricardo de Oliveira Schmidt, Ramin Sadre, Jeronimo A. Bezerra and Aiko Pras. *Assessing the quality of flow measurements from OpenFlow devices*. April 2016 (page 9).

[51] José Suárez-Varela and Pere Barlet-Ros. "Towards a NetFlow Implementation for OpenFlow Software-Defined Networks". In: *2017 29th International Teletraffic Congress (ITC 29)*. Vol. 1. September 2017, pp. 187–195 (page 9).

[52] Tomas Cejka, Vaclav Bartos, Lukas Truxa and Hana Kubatova. "Using Application-Aware Flow Monitoring for SIP Fraud Detection". In: *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 (page 16).

[53] Vojtech Krmicek, Jan Vykopal and Radek Krejci. "Netflow Based System for NAT Detection". In: *Proceedings of the 5th International Student Workshop on Emerging Networking Experiments and Technologies*. Co-Next Student Workshop '09. Rome, Italy: ACM, 2009, pp. 23–24. ISBN: 978-1-60558-751-6 (pages 16, 20).

[54] Byungjoon Lee, Hyeongu Son, Seunghyun Yoon and Youngseok Lee. "End-to-end Flow Monitoring with IPFIX". In: *Proceedings of the 10th Asia-Pacific Conference on Network Operations and Management Symposium: Managing Next Generation Networks and Services*. APNOMS'07. Sapporo, Japan: Springer-Verlag, 2007, pp. 225–234. ISBN: 978-3-540-75475-6 (page 16).

[55] Youngseok Lee, Seongho Shin, Soonbyoung Choi and Hyeon-gu Son. "IPv6 Anomaly Traffic Monitoring with IPFIX". In: *Proceedings of the Second International Conference on Internet Monitoring and Protection*. ICIMP '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 10–. ISBN: 0-7695-2911-9 (page 16).

[56] Maurizio Molina, Agostino Chiosi, Salvatore D'Antonio and Giorgio Ventre. "Design Principles and Algorithms for Effective High-speed IP Flow Monitoring". In: *Comput. Commun.* 29.10 (June 2006), pp. 1653–1664. ISSN: 0140-3664. DOI: 10.1016/j.comcom.2005.07.024 (pages 16, 22, 24, 114).

[57] Jian Zhang and Andrew Moore. "Traffic Trace Artifacts due to Monitoring Via Port Mirroring". In: *E2EMON*. IEEE Computer Society, 2007, pp. 1–8. ISBN: 1-4244-1289-7 (page 18).

[58] Sebastian Abt, Christian Dietz, Harald Baier and Slobodan Petrović. "Passive Remote Source NAT Detection Using Behavior Statistics Derived from Netflow". In: *Proceedings of the 7th IFIP WG 6.6 International Conference on Autonomous Infrastructure, Management, and Security: Emerging Management Mechanisms for the Future Internet - Volume 7943*. AIMS'13. Barcelona, Spain: Springer-Verlag, 2013, pp. 148–159. ISBN: 978-3-642-38997-9 (page 20).

[59] Cristian Estan, Ken Keys, David Moore and George Varghese. "Building a Better NetFlow". In: *SIGCOMM Comput. Commun. Rev.* 34.4 (August 2004), pp. 245–256. ISSN: 0146-4833. DOI: 10.1145/1030194.1015495 (pages 22, 149).

[60] Dawei Wang, Yibo Xue and Yingfei Dong. "Memory-Efficient Hypercube Flow Table for Packet Processing on Multi-Cores". In: *2011 IEEE Global Telecommunications Conference - GLOBECOM 2011*. December 2011, pp. 1–6 (pages 22, 113).

[61] Cisco Systems, Inc., San Jose, CA and USA. *Cisco IOS Flexible NetFlow Command Reference*. November 2013. URL: http://www.cisco.com/c/en/us/td/docs/ios/fnetflow/command/reference/fnf_book/fnf_01.html (Accessed on 21st June 2017) (page 23).

[62] Juan Moliná Rodriguez, Valentín Carela Español, Pere Barlet Ros, Ralf Hoffmann and Klaus Degner. "Empirical Analysis of Traffic to Establish a Profiled Flow Termination Timeout". In: *2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC)*. July 2013, pp. 1156–1161 (pages 23, 114).

[63] David Murray and Terrys Koziniec. "The State of Enterprise Network Traffic in 2012". In: *2012 18th Asia-Pacific Conference on Communications (APCC)*. October 2012, pp. 179–184 (page 24).

[64] Tsung-Huan Cheng, Ying-Dar Lin, Yuan-Cheng Lai and Po-Ching Lin. "Evasion Techniques: Sneaking through Your Intrusion Detection/Prevention Systems". In: *IEEE Communications Surveys Tutorials* 14.4 (October 2012), pp. 1011–1020. ISSN: 1553-877X. DOI: 10.1109/SURV.2011.092311.00082 (page 24).

[65] Michał Trojnara. *Stunnel*. July 2015. URL: https://www.stunnel.org/index.html (Accessed on 3rd August 2017) (page 29).

[66] Bingdong Li, Jeff Springer, George Bebis and Mehmet Hadi Gunes. "Review: A Survey of Network Flow Applications". In: *J. Netw. Comput. Appl.* 36.2 (March 2013), pp. 567–581. ISSN: 1084-8045. DOI: 10.1016/j.jnca.2012.12.020 (page 29).

[67] Muhammad Fahad Umer, Muhammad Sher and Yaxin Bi. "Flow-based intrusion detection: Techniques and challenges". In: *Computers & Security* 70 (2017), pp. 238–254. ISSN: 0167-4048. DOI: http://dx.doi.org/10.1016/j.cose.2017.05.009 (pages 29, 30).

[68] Timothy Shimeall, Sidney Faber, Markus DeShon and Andrew Kompanek. "Using SiLK for Network Traffic Analysis". In: *CERT R Network Situational Awareness Group, Carnegie Mellon University* (2014) (page 29).

[69] Peter Haag. *NFDUMP*. December 2014. URL: http://nfdump.sourceforge.net/ (Accessed on 3rd August 2017) (page 29).

[70] Rick Hofstede, Anna Sperotto, Tiago Fioreze and Aiko Pras. "The Network Data Handling War: MySQL vs. NfDump". In: *Networked Services and Applications - Engineering, Control and Management*. Vol. 6164. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, pp. 167–176 (page 29).

[71] Clinton Gormley and Zachary Tong. *Elasticsearch: The Definitive Guide*. 1st. O'Reilly Media, Inc., 2015. ISBN: 978-1449358549 (page 29).

[72] Yeonhee Lee and Youngseok Lee. "Toward Scalable Internet Traffic Measurement and Analysis with Hadoop". In: *SIGCOMM Comput. Commun. Rev.* 43.1 (January 2012), pp. 5–13. ISSN: 0146-4833. DOI: 10.1145/2427036.2427038 (page 29).

[73] Peter Haag. *NfSen*. December 2011. URL: http://nfsen.sourceforge.net/ (Accessed on 4th August 2017) (pages 30, 78, 90).

[74] Tomas Jirsik, Milan Cermak, Daniel Tovarnak and Pavel Celeda. "Toward Stream-Based IP Flow Analysis". In: *IEEE Communications Magazine* 55.7 (2017), pp. 70–76. ISSN: 0163-6804. DOI: 10.1109/MCOM.2017.1600972 (page 30).

[75] Milan Čermák, Daniel Tovarňák, Martin Laštovička and Pavel Čeleda. "A Performance Benchmark for NetFlow Data Analysis on Distributed Stream Processing Systems". In: *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*. April 2016, pp. 919–924 (page 30).

[76] Martin Laštovička and Pavel Čeleda. "Situational Awareness: Detecting Critical Dependencies and Devices in a Network". In: *Security of Networks and Services in an All-Connected World: 11th IFIP WG 6.6 International Conference on Autonomous Infrastructure, Management, and Security, AIMS 2017, Zurich, Switzerland, July 10-13, 2017, Proceedings*. Cham: Springer International Publishing, 2017, pp. 173–178. ISBN: 978-3-319-60774-0 (page 30).

[77] Varun Chandola, Arindam Banerjee and Vipin Kumar. "Anomaly Detection: A Survey". In: *ACM Comput. Surv.* 41.3 (July 2009), 15:1–15:58. ISSN: 0360-0300. DOI: 10.1145/1541880.1541882 (page 30).

[78] Nino Vincenzo Verde, Giuseppe Ateniese, Emanuele Gabrielli, Luigi Vincezo Mancini and Spognardi Angelo. "No NAT'd User Left Behind: Fingerprinting Users behind NAT from NetFlow Records Alone". In: *2014 IEEE 34th International Conference on Distributed Computing Systems*. June 2014, pp. 218–227 (page 30).

[79] Chih-Fong Tsai, Yu-Feng Hsu, Chia-Ying Lin and Wei-Yang Lin. "Intrusion detection by machine learning: A review". In: *Expert Systems with Applications* 36.10 (2009), pp. 11994–12000. ISSN: 0957-4174. DOI: http://dx.doi.org/10.1016/j.eswa.2009.05.029 (page 30).

[80] Robin Sommer and Vern Paxson. "Outside the Closed World: On Using Machine Learning for Network Intrusion Detection". In: *2010 IEEE Symposium on Security and Privacy*. May 2010, pp. 305–316 (page 30).

[81] Internet Assigned Numbers Authority. *IP Flow Information Export (IPFIX) Entities*. September 2017. URL: https://www.iana.org/assignments/ipfix/ipfix.xhtml (Accessed on 10th August 2017) (pages 33, 34).

[82] N. Spring, D. Wetherall and D. Ely. *Robust Explicit Congestion Notification (ECN) Signaling with Nonces*. RFC 3540 (Experimental). RFC. Fremont, CA, USA: RFC Editor, June 2003. URL: https://www.rfc-editor.org/rfc/rfc3540.txt (page 34).

[83] B. Trammell and P. Aitken. *Revision of the tcpControlBits IP Flow Information Export (IPFIX) Information Element*. RFC 7125 (Informational). RFC. Fremont, CA, USA: RFC Editor, February 2014. URL: https://www.rfc-editor.org/rfc/rfc7125.txt (page 34).

[84] Ming Gao, Kenong Zhang and Jiahua Lu. "Efficient packet matching for gigabit network intrusion detection using TCAMs". In: *Advanced Information Networking and Applications, 2006. AINA 2006. 20th International Conference on*. Vol. 1. 2006, pp. 1–6 (page 36).

[85] Haiguang Lai, Shengwen Cai, Hao Huang, Junyuan Xie and Hui Li. "A Parallel Intrusion Detection System for High-Speed Networks". In: *Applied Cryptography and Network Security*. Vol. 3089. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2004, pp. 439–451. ISBN: 978-3-540-22217-0 (page 36).

[86] Thuy T.T. Nguyen and Grenville Armitage. "A Survey of Techniques for Internet Traffic Classification Using Machine Learning". In: *Communications Surveys & Tutorials, IEEE* 10.4 (October 2008), pp. 56–76. ISSN: 1553-877X. DOI: 10.1109/SURV.2008.080406 (pages 36, 128).

[87] Alberto Dainotti, Antonio Pescape and Kimberly C. Claffy. "Issues and Future Directions in Traffic Classification". In: *Network, IEEE* 26.1 (January 2012), pp. 35–40. ISSN: 0890-8044. DOI: 10.1109/MNET.2012.6135854 (pages 36, 128).

[88] Michael Finsterbusch, Chris Richter, Eduardo Rocha, Jean-Alexander Muller and Klaus Hanssgen. "A Survey of Payload-Based Traffic Classification Approaches". In: *Communications Surveys & Tutorials, IEEE* 16.2 (May 2014), pp. 1135–1156. ISSN: 1553-877X. DOI: 10.1109/SURV.2013.100613.00161 (pages 36, 128, 137).

[89] Ruoming Pang, Vern Paxson, Robin Sommer and Larry Peterson. "binpac: A yacc for Writing Application Protocol Parsers". In: *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*. IMC '06. Rio de Janeriro, Brazil: ACM, 2006, pp. 289–300. ISBN: 1-59593-561-4 (pages 36, 44).

[90] Juan Caballero, Heng Yin, Zhenkai Liang and Dawn Song. "Polyglot: Automatic Extraction of Protocol Message Format Using Dynamic Binary Analysis". In: *Proceedings of the 14th ACM Conference on Computer and Communications Security*. CCS '07. Alexandria, Virginia, USA: ACM, 2007, pp. 317–329. ISBN: 978-1-59593-703-2 (page 36).

[91] Weidong Cui, Jayanthkumar Kannan and Helen J. Wang. "Discoverer: Automatic Protocol Reverse Engineering from Network Traces". In: *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*. SS'07. Boston, MA: USENIX Association, 2007, 14:1–14:14. ISBN: 111-333-5555-77-9 (page 36).

[92] Drew Davidson, Randy Smith, Nic Doyle and Somesh Jha. "Protocol Normalization Using Attribute Grammars". In: *Proceedings of the 14th European Conference on Research in Computer Security*. ESORICS'09. Saint-Malo, France: Springer-Verlag, 2009, pp. 216–231. ISBN: 978-3-642-04443-4 (page 36).

[93] Robin Sommer, Johanna Amann and Seth Hall. "Spicy: A Unified Deep Packet Inspection Framework for Safely Dissecting All Your Data". In: *Proceedings of the 32Nd Annual Conference on Computer Security Applications*. ACSAC '16. Los Angeles, California, USA: ACM, 2016, pp. 558–569. ISBN: 978-1-4503-4771-6 (page 37).

[94] Christopher M. Inacio and Brian Trammell. "YAF: Yet Another Flowmeter". In: *Proceedings of the 24th International Conference on Large Installation System Administration*. LISA'10. San Jose, CA: USENIX Association, 2010, pp. 1–16 (pages 37, 46).

[95] CERT Network Situational Awareness Group Engineering Team. *yaf application labeling*. URL: https://tools.netsa.cert.org/yaf/applabel.html (Accessed on 27th September 2017) (page 37).

[96] Emily Sarneso and the CERT Network Situational Awareness Group Engineering Team. *yaf deep packet inspection*. URL: https://tools.netsa.cert.org/yaf/yafdpi.html (Accessed on 27th September 2017) (page 37).

[97] Luca Deri. "nProbe: an Open Source NetFlow Probe for Gigabit Networks". In: *Proceedings of the TERENA Networking Conference, TNC'03*. Zagreb, Croatia, 2003 (page 37).

[98] Luca Deri, Maurizio Martinelli, Tomasz Bujlow and Alfredo Cardigliano. "nDPI: Open-source high-speed deep packet inspection". In: *Wireless Communications and Mobile Computing Conference (IWCMC), 2014 International*. Nicosia, Cyprus, August 2014, pp. 617–622 (pages 37, 137).

[99] Flowmon Networks. *Flowmon Probe*. URL: https://www.flowmon.com/en/products/flowmon/probe (Accessed on 27th September 2017) (pages 38, 45, 56, 68, 79, 117, 151).

[100] Shane Alcock and Richard Nelson. *Libprotoident: Traffic Classification Using Lightweight Packet Inspection*. Tech. rep. WAND Network Research Group, 2012. URL: http://www.wand.net.nz/~salcock/lpi/lpi.pdf (Accessed on 28th April 2017) (pages 38, 110, 137).

[101] Cisco Systems, Inc., San Jose, CA and USA. *Cisco Application Visibility and Control (AVC)*. URL: https://www.cisco.com/c/en/us/products/routers/avc-control.html (Accessed on 27th September 2017) (page 38).

[102] Cisco Systems, Inc., San Jose, CA and USA. *NBAR2 Protocol Library*. June 2013. URL: https://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/network-based-application-recognition-nbar/product_bulletin_c25-627831.html (Accessed on 27th September 2017) (page 38).

[103] Lancope. *Stealthwatch FlowSensor*. URL: https://www.lancope.com/products/stealthwatch-flowsensor (Accessed on 27th September 2017) (page 38).

[104] Palo Alto Networks. *Next-Generation Firewall*. URL: https://www.paloaltonetworks.com/products/secure-the-network/next-generation-firewall (Accessed on 27th September 2017) (page 38).

[105] Internet Assigned Numbers Authority. *Service Name and Transport Protocol Port Number Registry*. October 2017. URL: https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml (Accessed on 12th October 2017) (page 40).

[106] Tomasz Bujlow, Valentín Carela-Espanol and Pere Barlet-Ros. "Independent Comparison of Popular DPI Tools for Traffic Classification". In: *Computer Networks* 76.0 (2015), pp. 75–89. DOI: http://dx.doi.org/10.1016/j.comnet.2014.11.001 (pages 40, 137).

[107] GNU Project. *The GNU C Library (glibc)*. August 2017. URL: http://www.gnu.org/software/libc/ (Accessed on 18th October 2017) (page 43).

[108] Philip Hazel. *PCRE – Perl Compatible Regular Expressions*. 2015. URL: http://www.pcre.org/ (Accessed on 18th October 2017) (page 43).

[109] Open Information Security Foundation. *Suricata – network IDS, IPS and network security monitoring engine*. October 2017. URL: http://www.suricata-ids.org (Accessed on 18th October 2017) (page 43).

[110] Martin Roesch. "Snort - Lightweight Intrusion Detection for Networks". In: *Proceedings of the 13th USENIX conference on System administration*. LISA '99. Seattle, Washington: USENIX Association, 1999, pp. 229–238 (page 43).

[111] Qualys, Inc. *LibHTP – security-aware parser for the HTTP protocol*. July 2017. URL: http://github.com/ironbee/libhtp (Accessed on 18th October 2017) (page 43).

[112] Jason Bittel. *httpry - HTTP logging and information retrieval tool*. October 2014. URL: http://github.com/jbittel/httpry (Accessed on 18th October 2017) (page 43).

[113] Vern Paxson. "Bro: a system for detecting network intruders in real-time". In: *Comput. Netw.* 31.23-24 (December 1999), pp. 2435–2463. ISSN: 1389-1286. DOI: 10.1016/S1389-1286(99)00112-7 (page 44).

[114] John Levine and Levine John. *Flex & Bison*. 1st. O'Reilly Media, Inc., 2009. ISBN: 978-0596155971 (page 44).

[115] Mike E. Lesk and Eric Schmidt. *Lex – a Lexical Analyzer Generator*. Tech. rep. Computing Science Technical Report No. 39. Bell Laboratories, 1975 (page 44).

[116] Robert McNaughton and Hisao Yamada. "Regular Expressions and State Graphs for Automata". In: *Electronic Computers, IRE Transactions on* EC-9.1 (1960), pp. 39–47. ISSN: 0367-9950. DOI: 10.1109/TEC.1960.5221603 (page 44).

[117] Fabian Schneider, Sachin Agarwal, Tansu Alpcan and Anja Feldmann. "The new web: characterizing AJAX traffic". In: *Proceedings of the 9th international conference on Passive and active network measurement*. PAM'08. Cleveland, OH, USA: Springer-Verlag, 2008, pp. 31–40. ISBN: 978-3-540-79231-4 (page 44).

[118] Luis Miguel Torres, Eduardo Magana, Mikel Izal and Daniel Morato. "Identifying Sessions to Websites as an Aggregation of Related Flows". In: *Telecommunications Network Strategy and Planning Symposium (NETWORKS), 2012 XVth International*. 2012, pp. 1–6 (page 44).

[119] Luis Miguel Torres, Eduardo Magana, Mikel Izal and Daniel Morato. "Strategies for automatic labelling of web traffic traces". In: *37th Annual IEEE Conference on Local Computer Networks* 0 (2012), pp. 196–199. ISSN: 0742-1303. DOI: http://doi.ieeecomputersociety.org/10.1109/LCN.2012.6423605 (page 44).

[120] Vinicius Gehlen, Alessandro Finamore, Marco Mellia and Maurizio M. Munafò. "Uncovering the Big Players of the Web". In: *Proceedings of the 4th international conference on Traffic Monitoring and Analysis*. TMA'12. Vienna, Austria: Springer-Verlag, 2012, pp. 15–28. ISBN: 978-3-642-28533-2 (pages 44, 85).

[121] Aniket Mahanti, Carey Williamson, Niklas Carlsson, Martin Arlitt and Anirban Mahanti. "Characterizing the file hosting ecosystem: A view from the edge". In: *Perform. Eval.* 68.11 (November 2011), pp. 1085–1102. ISSN: 0166-5316. DOI: 10.1016/j.peva.2011.07.016 (page 44).

[122] R. Fielding and J. Reschke. *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing*. RFC 7230 (Proposed Standard). RFC. Fremont, CA, USA: RFC Editor, June 2014. URL: https://www.rfc-editor.org/rfc/rfc7230.txt (page 44).

[123] Tomáš Šíma, Petr Velan and Pavel Čeleda. *FlowMon – Plugins for HTTP Monitoring*. April 2013. URL: http://dior.ics.muni.cz/~velan/flowmon-input-http/ (Accessed on 18th October 2017) (page 45).

[124] Google. *Verifying Googlebot*. URL: https://support.google.com/webmasters/answer/80553 (Accessed on 20th October 2017) (page 55).

[125] Justin Billig, Yuri Danilchenko and Charles E. Frank. "Evaluation of Google Hacking". In: *Proceedings of the 5th Annual Conference on Information Security Curriculum Development*. InfoSecCD '08. Kennesaw, Georgia: ACM, 2008, pp. 27–32. ISBN: 978-1-60558-333-4 (pages 55, 64).

[126] Roberto Perdisci, Wenke Lee and Nick Feamster. "Behavioral Clustering of HTTP-based Malware and Signature Generation Using Malicious Network Traces". In: *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*. NSDI'10. San Jose, California: USENIX Association, 2010 (page 56).

[127] Martin Husák and Jakub Čegan. "PhiGARo: Automatic Phishing Detection and Incident Response Framework". In: *Availability, Reliability and Security (ARES), 2014 Ninth International Conference on*. 2014, pp. 295–302 (page 56).

[128] Brice Augustin and Abdelhamid Mellouk. "On Traffic Patterns of HTTP Applications". In: *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*. December 2011 (page 56).

[129] Guowu Xie, Marios Iliofotou, Thomas Karagiannis, Michalis Faloutsos and Yaohui Jin. "ReSurf: Reconstructing Web-Surfing Activity from Network Traffic". In: *IFIP Networking Conference, 2013*. May 2013 (page 56).

[130] Ye Xu, Gang Xiong, Yong Zhao and Li Guo. "Toward Identifying and Understanding User-Agent Strings in HTTP Traffic". English. In: *Web Technologies and Applications*. Vol. 8710. Lecture Notes in Computer Science. Springer International Publishing, 2014, pp. 177–187. ISBN: 978-3-319-11118-6 (page 56).

[131] Jaroslav Mallat. *UASparser*. 2017. URL: `http://user-agent-string.info/download/UASparser` (Accessed on 20th October 2017) (page 56).

[132] Chang-Gyu Jin and Mi-Jung Choi. "Integrated Analysis Method on HTTP Traffic". In: *Network Operations and Management Symposium (APNOMS), 2012 14th Asia-Pacific*. September 2012 (page 56).

[133] Min Hur and Myung-Sup Kim. "Towards Smart Phone Traffic Classification". In: *Network Operations and Management Symposium (APNOMS), 2012 14th Asia-Pacific*. September 2012 (page 56).

[134] Niels Provos and Thorsten Holz. *Virtual Honeypots: From Botnet Tracking to Intrusion Detection*. First. Addison-Wesley Professional, 2007. ISBN: 9780321336323 (page 56).

[135] Monowar H. Bhuyan, D.K. Bhattacharyya and J.K. Kalita. "Surveying Port Scans and Their Detection Methodologies". In: *Comput. J.* 54.10 (October 2011), pp. 1565–1581. ISSN: 0010-4620 (page 60).

[136] Giovane César Moreira Moura. "Internet Bad Neighborhoods". PhD thesis. Enschede: University of Twente, March 2013. URL: `http://doc.utwente.nl/84507/` (page 64).

[137] WordPress Codex. *Brute Force Attacks*. URL: `http://codex.wordpress.org/Brute_Force_Attacks` (Accessed on 20th October 2017) (page 64).

[138] Jan Vykopal, Martin Drašar and Philipp Winter. "Flow-based Brute-force Attack Detection". eng. In: *Advances in IT Early Warning*. Ed. by Eckehard Hermann Markus Zeilinger Peter Schoo. Stuttgart: Fraunhofer Verlag, 2013, pp. 41–51. ISBN: 978-3-8396-0474-8 (pages 64, 65).

[139] Anna Sperotto, Gregor Schaffrath, Ramin Sadre, Cristian Morariu, Aiko Pras and Burkhard Stiller. "An Overview of IP Flow-Based Intrusion Detection". In: *Communications Surveys Tutorials, IEEE* 12.3 (April 2010), pp. 343–356. ISSN: 1553-877X (page 65).

[140] MinGyung Kang, Juan Caballero and Dawn Song. "Distributed Evasive Scan Techniques and Countermeasures". English. In: *Detection of Intrusions and Malware, and Vulnerability Assessment*. Vol. 4579. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2007, pp. 157–174. ISBN: 978-3-540-73613-4 (page 65).

[141] Yang Sun, Isaac G. Councill and C. Lee Giles. "The Ethicality of Web Crawlers". In: *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2010 IEEE/WIC/ACM International Conference on*. Vol. 1. August 2010, pp. 668–675 (page 65).

[142] Oliver Hohlfeld, Thomas Graf and Florin Ciucu. "Longtime Behavior of Harvesting Spam Bots". In: *Proceedings of the 2012 ACM Conference on Internet Measurement Conference*. IMC '12. Boston, Massachusetts, USA: ACM, 2012, pp. 453–460. ISBN: 978-1-4503-1705-4 (page 65).

[143] Kimberly C. Claffy. "Tracking IPv6 Evolution: Data We Have and Data We Need". In: *SIGCOMM Comput. Commun. Rev.* 41.3 (July 2011), pp. 43–48. ISSN: 0146-4833. DOI: `10.1145/2002250.2002258` (page 66).

[144] Mohammad Aazam, Imran Khan, Muhammad Alam and Amir Qayyum. "Comparison of IPv6 Tunneled Traffic of Teredo and ISATAP over Test-bed Setup". In: *2010 International Conference on Information and Emerging Technologies*. June 2010, pp. 1–4 (page 67).

[145] Sebastian Zander, Lachlan L.H. Andrew, Grenville Armitage, Geoff Huston and George Michaelson. "Investigating the IPv6 Teredo Tunnelling Capability and Performance of Internet Clients". In: *SIGCOMM Comput. Commun. Rev.* 42.5 (September 2012), pp. 13–20. ISSN: 0146-4833. DOI: `10.1145/2378956.2378959` (page 67).

[146] Nazrulazhar Bahaman, Erman Hamid and Anton Satria Prabuwono. "Network performance evaluation of 6to4 tunneling". In: *2012 International Conference on Innovation Management and Technology Research*. May 2012, pp. 263–268 (page 67).

[147] S. Krishnan, D. Thaler and J. Hoagland. *Security Concerns with IP Tunneling*. RFC 6169 (Informational). RFC. Fremont, CA, USA: RFC Editor, April 2011. URL: https://www.rfc-editor.org/rfc/rfc6169.txt (page 67).

[148] Nadi Sarrar, Gregor Maier, Bernhard Ager, Robin Sommer and Steve Uhlig. "Investigating IPv6 Traffic: What Happened at the World IPv6 Day?" In: *Proceedings of the 13th International Conference on Passive and Active Measurement*. PAM'12. Vienna, Austria: Springer-Verlag, 2012, pp. 11–20. ISBN: 978-3-642-28536-3 (page 67).

[149] C. Huitema. *Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs)*. RFC 4380 (Proposed Standard). RFC. Updated by RFCs 5991, 6081. Fremont, CA, USA: RFC Editor, February 2006. URL: https://www.rfc-editor.org/rfc/rfc4380.txt (page 68).

[150] B. Carpenter and K. Moore. *Connection of IPv6 Domains via IPv4 Clouds*. RFC 3056 (Proposed Standard). RFC. Fremont, CA, USA: RFC Editor, February 2001. URL: https://www.rfc-editor.org/rfc/rfc3056.txt (page 68).

[151] Martin Elich, Matěj Grégr and Pavel Čeleda. "Monitoring of Tunneled IPv6 Traffic Using Packet Decapsulation and IPFIX". In: *Proceedings of the Third International Conference on Traffic Monitoring and Analysis*. TMA'11. Vienna, Austria: Springer-Verlag, 2011, pp. 64–71. ISBN: 978-3-642-20304-6 (pages 68, 76).

[152] Martin Elich, Petr Velan and Pavel Čeleda. *FlowMon IPv6 Tunnel Monitoring Plugin*. May 2013. URL: http://dior.ics.muni.cz/~velan/flowmon-input-ip6tun/ (Accessed on 25th October 2017) (page 68).

[153] MaxMind, Inc. *MaxMind GeoIP services*. 2013. URL: http://www.maxmind.com (page 70).

[154] Noah Davids. *Initial TTL values*. April 2017. URL: http://noahdavids.org/self_published/TTL_values.html (Accessed on 25th October 2017) (page 70).

[155] Cebrail Çiflikli, Ali Gerzer and Abdulah Tuncay Özşahin. "Packet traffic features of IPv6 and IPv4 protocol traffic". In: *Turkish Journal of Electrical Engineering & Computer Sciences* 20.5 (2012), pp. 727–749. DOI: 10.3906/elk-1008-696 (page 74).

[156] T. Narten, E. Nordmark, W. Simpson and H. Soliman. *Neighbor Discovery for IP version 6 (IPv6)*. RFC 4861 (Draft Standard). RFC. Updated by RFCs 5942, 6980, 7048, 7527, 7559, 8028. Fremont, CA, USA: RFC Editor, September 2007. URL: https://www.rfc-editor.org/rfc/rfc4861.txt (page 75).

[157] Ethan Katz-Bassett, John P. John, Arvind Krishnamurthy, David Wetherall, Thomas Anderson and Yatin Chawathe. "Towards IP Geolocation Using Delay and Topology Measurements". In: *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement (IMC)*. ACM, 2006, pp. 71–84 (page 78).

[158] Brian Eriksson, Paul Barford, Joel Sommers and Robert Nowak. "A Learning-Based Approach for IP Geolocation". In: *Proceedings of the 11th International Conference on Passive and Active Measurement (PAM)*. Lecture Notes in Computer Science. 2010, pp. 171–180 (page 78).

[159] Ingmar Poese, Steve Uhlig, Mohamed Ali Kaafar, Benoit Donnet and Bamba Gueye. "IP Geolocation Databases: Unreliable?" In: *SIGCOMM Comput. Commun. Rev.* 41.2 (April 2011), pp. 53–56. ISSN: 0146-4833. DOI: 10.1145/1971162.1971171 (pages 78, 80).

[160] geoPlugin. *geoPlugin to geolocate your visitors*. URL: http://www.geoplugin.com (Accessed on 25th October 2017) (page 78).

[161] MaxMind. *GeoLite Databases*. URL: http://www.maxmind.com/app/geoip_country (Accessed on 25th October 2017) (pages 78, 79).

[162] IP2Location. *IP Address Geolocation to Identify Website Visitor's Geographical Location*. URL: http://www.ip2location.com (Accessed on 25th October 2017) (page 78).

[163] ntop. *nProbe*. URL: http://www.ntop.org/products/nprobe/ (Accessed on 25th October 2017) (page 79).

[164] CERT Network Situational Awareness Team (NetSA). *SiLK*. URL: http://tools.netsa.cert.org/silk/ (Accessed on 13th December 2012) (page 79).

[165] ntop. *ntop*. URL: http://www.ntop.org/products/ntop/ (Accessed on 25th October 2017) (page 79).

[166] QoSient. *ARGUS - Auditing Network Activity*. URL: http://www.qosient.com/argus/argusnetflow.shtml (Accessed on 25th October 2017) (page 79).

[167] Rick Hofstede. *SURFmap: A Network Monitoring Tool Based on the Google Maps API*. URL: http://surfmap.sourceforge.net/ (Accessed on 25th October 2017) (page 79).

[168] Rick Hofstede and Tiago Fioreze. "SURFmap: A Network Monitoring Tool Based on the Google Maps API". In: *IFIP/IEEE International Symposium on Integrated Network Management (IM)*. June 2009, pp. 676–690 (page 79).

[169] Raphael Thomas Blatter. "Extending HAPviewer: Time Window, Flow Classification, and Geolocation". MA thesis. University of Zurich, Department of Informatics, 2011. URL: ftp://ftp.tik.ee.ethz.ch:21/pub/students/2011-FS/MA-2011-12.pdf (page 79).

[170] Olivier Festor and Abdelkader Lahmadi. *Information Elements for device location in IPFIX*. Internet-Draft. July 2012 (page 80).

[171] G.T. Matthijs van Polen, Giovane César Moreira Moura and Aiko Pras. "Finding and Analyzing Evil Cities on the Internet". In: *Proceedings of the 5th International Conference on Autonomous Infrastructure, Management and Security (AIMS)*. Vol. 6734. Notes in Computer Science. 2011, pp. 38–48 (page 83).

[172] Andrew Moore, Michael Crogan and Denis Zuev. *Discriminators for use in flow-based classification*. Tech. rep. Queen Mary, University of London, 2005. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.101.7450&rep=rep1&type=pdf (pages 86, 138).

[173] Will E. Leland, Murad S. Taqqu, Walter Willinger and Daniel V. Wilson. "On the Self-similar Nature of Ethernet Traffic". In: *Conference Proceedings on Communications Architectures, Protocols and Applications*. SIGCOMM '93. San Francisco, California, USA: ACM, 1993, pp. 183–193. ISBN: 0-89791-619-0 (page 87).

[174] Mark E. Crovella and Azer Bestavros. "Self-similarity in World Wide Web traffic: evidence and possible causes". In: *Networking, IEEE/ACM Transactions on* 5.6 (December 1997), pp. 835–846. ISSN: 1063-6692. DOI: 10.1109/90.650143 (page 87).

[175] Vern Paxson and Sally Floyd. "Wide Area Traffic: The Failure of Poisson Modeling". In: *IEEE/ACM Trans. Netw.* 3.3 (June 1995), pp. 226–244. ISSN: 1063-6692. DOI: 10.1109/90.392383 (page 87).

[176] Chuck Fraleigh, Sue Moon, Bryan Lyles, Chase Cotton, Mujahid Khan, Deb Moll, Rob Rockell, Ted Seely and Sprint Christophe Diot. "Packet-level traffic measurements from the Sprint IP backbone". In: *Network, IEEE* 17.6 (November 2003), pp. 6–16. ISSN: 0890-8044. DOI: 10.1109/MNET.2003.1248656 (page 87).

[177] Marina Fomenkov, Ken Keys, David Moore and K Claffy. "Longitudinal Study of Internet Traffic in 1998-2003". In: *Proceedings of the Winter International Synposium on Information and Communication Technologies*. WISICT '04. Cancun, Mexico: Trinity College Dublin, 2004, pp. 1–6 (page 87).

[178] Wolfgang John and Sven Tafvelin. "Analysis of Internet Backbone Traffic and Header Anomalies Observed". In: *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*. IMC '07. San Diego, California, USA: ACM, 2007, pp. 111–116. ISBN: 978-1-59593-908-1 (page 87).

[179] Theophilus Benson, Aditya Akella and David A. Maltz. "Network Traffic Characteristics of Data Centers in the Wild". In: *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*. IMC '10. Melbourne, Australia: ACM, 2010, pp. 267–280. ISBN: 978-1-4503-0483-2 (page 88).

[180] José Luis García-Dorado, Alessandro Finamore, Marco Mellia, Michela Meo and Maurizio M. Munafo. "Characterization of ISP Traffic: Trends, User Habits, and Access Technology Impact". In: *Network and Service Management, IEEE Transactions on* 9.2 (June 2012), pp. 142–155. ISSN: 1932-4537. DOI: 10.1109/TNSM.2012.022412.110184 (page 88).

[181] Lin Quan and John Heidemann. "On the Characteristics and Reasons of Long-lived Internet Flows". In: *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*. IMC '10. Melbourne, Australia: ACM, 2010, pp. 444–450. ISBN: 978-1-4503-0483-2 (page 91).

[182] Andrew W. Moore and Konstantina Papagiannaki. "Toward the Accurate Identification of Network Applications". In: *Proceedings of the 6th International Conference on Passive and Active Network Measurement*. PAM'05. Boston, MA: Springer-Verlag, 2005, pp. 41–54. ISBN: 978-3-540-25520-8 (page 95).

[183] Intel Corporation. *An Introduction to the Intel® QuickPath Interconnect*. January 2009. URL: https://www.intel.com/content/dam/doc/white-paper/quick-path-interconnect-introduction-paper.pdf (Accessed on 15th November 2017) (pages 101, 118).

[184] José Luis García-Dorado, Felipe Mata, Javier Ramos, Pedro M. Santiago del Río, Victor Moreno and Javier Aracil. "High-Performance Network Traffic Processing Systems Using Commodity Hardware". English. In: *Data Traffic Monitoring and Analysis*. Vol. 7754. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pp. 3–27. ISBN: 978-3-642-36783-0 (pages 101, 106).

[185] Georges Nassopulos, Dario Rossi, Francesco Gringoli, Lorenzo Nava, Maurizio Dusi and Pedro Maria Santiago del Rio. "Flow Management at Multi-Gbps: Tradeoffs and Lessons Learned". In: *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 (pages 101, 113).

[186] The Internet Engineering Steering Group. *IP Performance Measurement (ippm)*. October 1997. URL: https://datatracker.ietf.org/wg/ippm/ (Accessed on 29th November 2017) (page 101).

[187] The Internet Engineering Steering Group. *Benchmarking Methodology (bmwg)*. October 1989. URL: https://datatracker.ietf.org/wg/bmwg/ (Accessed on 9th February 2018) (page 101).

[188] V. Paxson, G. Almes, J. Mahdavi and M. Mathis. *Framework for IP Performance Metrics*. RFC 2330 (Informational). RFC. Updated by RFC 7312. Fremont, CA, USA: RFC Editor, May 1998. URL: https://www.rfc-editor.org/rfc/rfc2330.txt (page 101).

[189] A. Morton. *IMIX Genome: Specification of Variable Packet Sizes for Additional Testing*. RFC 6985 (Informational). RFC. Fremont, CA, USA: RFC Editor, July 2013. URL: https://www.rfc-editor.org/rfc/rfc6985.txt (page 101).

[190] NETCOPE Technologies. *Modelling of High Bandwidth Systems*. August 2017. URL: https://www.netcope.com/getattachment/ecf0b75f-b961-485a-94e1-ee6e1bd7bef4/Modelling-of-High-Bandwidth-Systems.aspx (Accessed on 30th November 2017) (page 102).

[191] Sebastian Gallenmüller, Paul Emmerich, Florian Wohlfart, Daniel Raumer and Georg Carle. "Comparison of Frameworks for High-Performance Packet IO". In: *Proceedings of the Eleventh ACM/IEEE Symposium on Architectures for Networking and Communications Systems*. ANCS '15. Oakland, California, USA: IEEE Computer Society, 2015, pp. 29–38. ISBN: 978-1-4673-6632-8 (pages 102, 106, 112).

[192] Samuel Williams, Andrew Waterman and David Patterson. "Roofline: An Insightful Visual Performance Model for Multicore Architectures". In: *Commun. ACM* 52.4 (April 2009), pp. 65–76. ISSN: 0001-0782. DOI: 10.1145/1498765.1498785 (page 102).

[193] Ryota Kawashima, Shin Muramatsu, Hiroki Nakayama, Tsunemasa Hayashi and Hiroshi Matsuo. "A Host-Based Performance Comparison of 40G NFV Environments Focusing on Packet Processing Architectures and Virtual Switches". In: *2016 Fifth European Workshop on Software-Defined Networks (EWSDN)*. October 2016, pp. 19–24 (page 102).

[194] Inc. Linux Kernel Organization. *PACKET_MMAP*. November 2017. URL: https://www.kernel.org/doc/Documentation/networking/packet%5C_mmap.txt (Accessed on 9th February 2018) (page 104).

[195] Loris Degioanni, Mario Baldi, Fulvio Risso and Gianluca Varenni. "Profiling and Optimization of Software-Based Network-Analysis Applications". In: *Proceedings of the 15th Symposium on Computer Architecture and High Performance Computing*. SBAC-PAD '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 226–. ISBN: 0-7695-2046-4 (page 104).

[196] Luca Deri. "Improving Passive Packet Capture: Beyond Device Polling". In: *Proceedings of SANE*. Vol. 2004. Amsterdam, Netherlands. 2004, pp. 85–93 (pages 104, 109).

[197] University of Waikato. *The Dag Project*. URL: http://dag.cs.waikato.ac.nz/ (Accessed on 10th February 2018) (page 104).

[198] Endace Technology Limited. *Endace Homepage*. URL: https://www.endace.com/ (Accessed on 10th February 2018) (page 104).

[199] Loris Degioanni and Gianluca Varenni. "Introducing Scalability in Network Measurement: Toward 10 Gbps with Commodity Hardware". In: *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*. IMC '04. Taormina, Sicily, Italy: ACM, 2004, pp. 233–238. ISBN: 1-58113-821-0 (page 104).

[200] Luca Deri. "nCap: Wire-speed Packet Capture and Transmission". In: *Workshop on End-to-End Monitoring Techniques and Services, 2005.* May 2005, pp. 47–55 (page 104).

[201] ntop. *PF_RING DNA*. February 2010. URL: https://www.ntop.org/pf_ring/introducing-pf_ring-dna-direct-nic-access/ (Accessed on 12th February 2018) (page 105).

[202] Francesco Fusco and Luca Deri. "High Speed Network Traffic Analysis with Commodity Multi-core Systems". In: *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*. IMC '10. Melbourne, Australia: ACM, 2010, pp. 218–224. ISBN: 978-1-4503-0483-2 (page 105).

[203] ntop. *PF_RING Zero Copy*. URL: https://www.ntop.org/products/packet-capture/pf_ring/pf_ring-zc-zero-copy/ (Accessed on 13th February 2018) (page 105).

[204] Brendan Gregg. *KPTI/KAISER Meltdown Initial Performance Regressions*. February 2018. URL: http://www.brendangregg.com/blog/2018-02-09/kpti-kaiser-meltdown-performance.html (Accessed on 13th February 2018) (page 105).

[205] Nicola Bonelli, Andrea Di Pietro, Stefano Giordano and Gregorio Procissi. "On Multi—gigabit Packet Capturing with Multi—core Commodity Hardware". In: *Proceedings of the 13th International Conference on Passive and Active Measurement*. PAM'12. Vienna, Austria: Springer-Verlag, 2012, pp. 64–73. ISBN: 978-3-642-28536-3 (page 105).

[206] Nicola Bonelli, Stefano Giordano, Gregorio Procissi and Luca Abeni. "A Purely Functional Approach to Packet Processing". In: *Proceedings of the Tenth ACM/IEEE Symposium on Architectures for Networking and Communications Systems*. ANCS '14. Los Angeles, California, USA: ACM, 2014, pp. 219–230. ISBN: 978-1-4503-2839-5 (page 105).

[207] Nicola Bonelli, Stefano Giordano and Gregorio Procissi. "Network Traffic Processing With PFQ". In: *IEEE Journal on Selected Areas in Communications* 34.6 (June 2016), pp. 1819–1833. ISSN: 0733-8716. DOI: 10.1109/JSAC.2016.2558998 (page 105).

[208] Nicola Bonelli, Stefano Giordano and Gregorio Procissi. "Enabling Packet Fan-Out in the libpcap Library for Parallel Traffic Processing". In: *2017 Network Traffic Measurement and Analysis Conference (TMA)*. June 2017, pp. 1–9 (page 105).

[209] Luigi Rizzo. "Netmap: a novel framework for fast packet I/O". In: *21st USENIX Security Symposium (USENIX Security 12)*. 2012, pp. 101–112 (page 105).

[210] Linux Foundation Project. *Data Plane Development Kit*. URL: https://dpdk.org (Accessed on 13th February 2018) (page 106).

[211] Linux Foundation Project. *DPDK Supported NICs*. URL: http://dpdk.org/doc/nics (Accessed on 13th February 2018) (page 106).

[212] Kieran Mansley, Greg Law, David Riddoch, Guido Barzini, Neil Turton and Steven Pope. "Getting 10 Gb/s from Xen: Safe and Fast Device Access from Unprivileged Domains". In: *Euro-Par 2007 Workshops: Parallel Processing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 224–233. ISBN: 978-3-540-78474-6 (page 106).

[213] Sangjin Han, Keon Jang, KyoungSoo Park and Sue Moon. "PacketShader: A GPU-accelerated Software Router". In: *Proceedings of the ACM SIGCOMM 2010 Conference*. SIGCOMM '10. New Delhi, India: ACM, 2010, pp. 195–206. ISBN: 978-1-4503-0201-2 (page 106).

[214] Lothar Braun, Alexander Didebulidze, Nils Kammenhuber and Georg Carle. "Comparing and Improving Current Packet Capturing Solutions Based on Commodity Hardware". In: *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*. IMC '10. Melbourne, Australia: ACM, 2010, pp. 206–217. ISBN: 978-1-4503-0483-2 (page 106).

[215] Haipeng Wang, Dazhong He and Huan Wang. "Comparison of High-Performance Packet Processing Frameworks on NUMA". In: *2016 7th IEEE International Conference on Software Engineering and Service Science (ICSESS)*. August 2016, pp. 54–58 (page 106).

[216] Tom Barbette, Cyril Soldani and Laurent Mathy. "Fast Userspace Packet Processing". In: *Proceedings of the Eleventh ACM/IEEE Symposium on Architectures for Networking and Communications Systems*. ANCS '15. Oakland, California, USA: IEEE Computer Society, 2015, pp. 5–16. ISBN: 978-1-4673-6632-8 (page 106).

[217] Intel Corporation. *Intel Ethernet Converged Network Adapters XL710 10/40 GbE*. 2014. URL: http://www.intel.com/content/www/us/en/network-adapters/converged-network-adapters/ethernet-xl710.html (Accessed on 21st February 2018) (page 107).

[218] Luca Deri, Alfredo Cardigliano and Francesco Fusco. "10 Gbit Line Rate Packet-to-disk Using n2disk". In: *Computer Communications Workshops (INFOCOM WKSHPS), 2013 IEEE Conference on*. April 2013, pp. 441–446 (page 107).

[219] Gianni Antichi, Stefano Giordano, David J. Miller and Andrew W. Moore. "Enabling open-source high speed network monitoring on NetFPGA". In: *Network Operations and Management Symposium (NOMS), 2012 IEEE*. April 2012, pp. 1029–1035 (page 107).

[220] Liberouter. *Hanic Design*. URL: https://www.liberouter.org/technologies/hanic/ (Accessed on 19th February 2018) (page 108).

[221] Shinae Woo and KyoungSoo Park. *Scalable TCP session monitoring with symmetric receive-side scaling*. Tech. rep. 2012. URL: https://an.kaist.ac.kr/~shinae/paper/2012-srss.pdf (page 108).

[222] Martin Zadnik, Jan Korenek, Petr Kobiersky and Ondrej Lengal. "Network Probe for Flexible Flow Monitoring". In: *2008 11th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems*. April 2008, pp. 1–6 (page 109).

[223] Lukas Kekely, Jan Kucera, Viktor Pus, Jan Korenek and Athanasios V. Vasilakos. "Software Defined Monitoring of Application Protocols". In: *IEEE Trans. Comput.* 65.2 (February 2016), pp. 615–626. ISSN: 0018-9340. DOI: 10.1109/TC.2015.2423668 (pages 110, 160).

[224] Haoyu Song and John W. Lockwood. "Efficient Packet Classification for Network Intrusion Detection Using FPGA". In: *Proceedings of the 2005 ACM/SIGDA 13th International Symposium on Field-programmable Gate Arrays*. FPGA '05. Monterey, California, USA: ACM, 2005, pp. 238–245. ISBN: 1-59593-029-9 (page 110).

[225] Adrienne Porter Felt, Richard Barnes, April King, Chris Palmer, Chris Bentzel and Parisa Tabriz. "Measuring HTTPS Adoption on the Web". In: *26th USENIX Security Symposium*. 2017, pp. 1323–1338 (page 113).

[226] Ramin Sadre, Anna Sperotto and Aiko Pras. "The Effects of DDoS Attacks on Flow Monitoring Applications". In: *2012 IEEE Network Operations and Management Symposium*. April 2012, pp. 269–277 (pages 113, 114).

[227] Liberouter. *COMBO-80G*. URL: https://www.liberouter.org/combo-80g/ (Accessed on 21st February 2018) (page 116).

[228] Sandvine, Inc. *Global Internet Phenomena Report 1H 2014*. May 2014. URL: https://www.sandvine.com/downloads/general/global-internet-phenomena/2014/1h-2014-global-internet-phenomena-report.pdf (Accessed on 25th November 2014) (page 128).

[229] Min Zhang, Wolfgang John, Kimberly C. Claffy and Nevil Brownlee. "State of the Art in Traffic Classification: A Research Review". In: *PAM '09: 10th International Conference on Passive and Active Measurement, Student Workshop*. Seoul, Korea, 2009 (page 128).

[230] Arthur Callado, Carlos Kamienski, Geza Szabo, Balazs Peter Gero, Judith Kelner, Stenio Fernandes and Djamel Sadok. "A Survey on Internet Traffic Identification". In: *Communications Surveys & Tutorials, IEEE* 11.3 (August 2009), pp. 37–52. ISSN: 1553-877X. DOI: 10.1109/SURV.2009.090304 (page 128).

[231] Zigang Cao, Gang Xiong, Yong Zhao, Zhenzhen Li and Li Guo. "A Survey on Encrypted Traffic Classification". English. In: *Applications and Techniques in Information Security*. Vol. 490. Communications in Computer and Information Science. Springer Berlin Heidelberg, 2014, pp. 73–81. ISBN: 978-3-662-45669-9 (page 128).

[232] Jawad Khalife, Amjad Hajjar and Jesus Diaz-Verdejo. "A multilevel taxonomy and requirements for an optimal traffic-classification model". In: *International Journal of Network Management* 24.2 (2014), pp. 101–120. ISSN: 1099-1190. DOI: 10.1002/nem.1855 (pages 128, 134, 135, 145).

[233] ISO. *ISO/IEC 7498-1:1994 Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model*. 2nd ed. Geneva, Switzerland: International Organization for Standardization, November 1994. URL: http://www.iso.org/iso/catalogue_detail.htm?csnumber=20269 (page 129).

[234] Sheila Frankel, Karen Kent, Ryan Lewkowski, Angela D. Orebaugh, Ronald W. Ritchey and Steven R. Sharma. *SP 800-77. Guide to IPsec VPNs*. Tech. rep. Gaithersburg, MD, United States: National Institute of Standards & Technology, 2005. URL: http://csrc.nist.gov/publications/nistpubs/800-77/sp800-77.pdf (page 129).

[235] C. Kaufman, P. Hoffman, Y. Nir and P. Eronen. *Internet Key Exchange Protocol Version 2 (IKEv2)*. RFC 5996 (Proposed Standard). RFC. Obsoleted by RFC 7296, updated by RFCs 5998, 6989. Fremont, CA, USA: RFC Editor, September 2010. URL: https://www.rfc-editor.org/rfc/rfc5996.txt (page 129).

[236] S. Kent. *IP Encapsulating Security Payload (ESP)*. RFC 4303 (Proposed Standard). RFC. Fremont, CA, USA: RFC Editor, December 2005. URL: https://www.rfc-editor.org/rfc/rfc4303.txt (page 129).

[237] T. Dierks and E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.2*. RFC 5246 (Proposed Standard). RFC. Updated by RFCs 5746, 5878, 6176, 7465, 7507, 7568, 7627, 7685, 7905, 7919. Fremont, CA, USA: RFC Editor, August 2008. URL: https://www.rfc-editor.org/rfc/rfc5246.txt (page 130).

[238] A. Freier, P. Karlton and P. Kocher. *The Secure Sockets Layer (SSL) Protocol Version 3.0*. RFC 6101 (Historic). RFC. Fremont, CA, USA: RFC Editor, August 2011. URL: https://www.rfc-editor.org/rfc/rfc6101.txt (page 130).

[239] Christopher Meyer. "20 Years of SSL/TLS Research An Analysis of the Internet's Security Foundation". PhD thesis. Ruhr-University Bochum, February 2014. URL: http://www-brs.ub.ruhr-uni-bochum.de/netahtml/HSS/Diss/MeyerChristopher/diss.pdf (page 130).

[240] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley and W. Polk. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. RFC 5280 (Proposed Standard). RFC. Updated by RFC 6818. Fremont, CA, USA: RFC Editor, May 2008. URL: https://www.rfc-editor.org/rfc/rfc5280.txt (pages 130, 133).

[241] T. Ylonen and C. Lonvick. *The Secure Shell (SSH) Transport Layer Protocol*. RFC 4253 (Proposed Standard). RFC. Updated by RFC 6668. Fremont, CA, USA: RFC Editor, January 2006. URL: https://www.rfc-editor.org/rfc/rfc4253.txt (pages 131, 133).

[242] J. Galbraith and O. Saarenmaa. *SSH File Transfer Protocol draft-ietf-secsh-filexfer-13.txt*. Internet-Draft. July 2006. URL: https://tools.ietf.org/html/draft-ietf-secsh-filexfer-13 (page 131).

[243] Jan Pechanec. *How the SCP protocol works. Jan Pechanec's Weblog*. July 2007. URL: https://blogs.oracle.com/janp/entry/how_the_scp_protocol_works (Accessed on 30th October 2014) (page 131).

[244] William Stallings. "Protocol Basics: Secure Shell Protocol". In: *The Internet Protocol Journal* 12.4 (December 2009), pp. 18–30 (page 131).

[245] David Harrison. *Index of BitTorrent Enhancement Proposals*. October 2014. URL: http://www.bittorrent.org/beps/bep_0000.html (Accessed on 27th November 2014) (page 131).

[246] Azureus Software Inc. *Message Stream Encryption. Vuze Wiki*. May 2014. URL: http://wiki.vuze.com/w/Message_Stream_Encryption (Accessed on 31st October 2014) (page 131).

[247] Skype and Microsoft. *Skype*. 2014. URL: http://www.skype.com/ (Accessed on 25th November 2014) (page 132).

[248] Davide Adami, Christian Callegar, Stefano Giordano, Michele Pagano and Teresa Pepe. "Skype-Hunter: A real-time system for the detection and classification of Skype traffic". In: *International Journal of Communication Systems* 25.3 (February 2011), pp. 386–403. DOI: 10.1002/dac.1247 (page 132).

[249] Skype Limited. *Skype Connect™ Requirements Guide*. 2011. URL: http://download.skype.com/share/business/guides/skype-connect-requirements-guide.pdf (Accessed on 1st November 2014) (page 132).

[250] IANA - Internet Assigned Numbers Authority. *Protocol Registries*. 2014. URL: http://www.iana.org/protocols (Accessed on 28th November 2014) (page 133).

[251] Qualys, Inc. *HTTP Client Fingerprinting Using SSL Handshake Analysis*. 2014. URL: https://www.ssllabs.com/projects/client-fingerprinting/ (Accessed on 28th November 2014) (page 133).

[252] Marek Majkowski. *SSL fingerprinting for p0f*. June 2012. URL: https://idea.popcount.org/2012-06-17-ssl-fingerprinting-for-p0f/ (Accessed on 28th November 2014) (page 133).

[253] Ralph Holz, Lothar Braun, Nils Kammenhuber and Georg Carle. "The SSL Landscape: A Thorough Analysis of the X.509 PKI Using Active and Passive Measurements". In: *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*. IMC '11. Berlin, Germany: ACM, 2011, pp. 427–444. ISBN: 978-1-4503-1013-0 (page 133).

[254] Zakir Durumeric, James Kasten, Michael Bailey and J. Alex Halderman. "Analysis of the HTTPS Certificate Ecosystem". In: *Proceedings of the 2013 Conference on Internet Measurement Conference*. IMC '13. Barcelona, Spain: ACM, 2013, pp. 291–304. ISBN: 978-1-4503-1953-9 (page 133).

[255] sslbl.abuse.ch. *SSL Blacklist*. 2014. URL: https://sslbl.abuse.ch/ (Accessed on 28th November 2014) (page 134).

[256] S. Blake-Wilson, M. Nystrom, D. Hopwood, J. Mikkelsen and T. Wright. *Transport Layer Security (TLS) Extensions*. RFC 4366 (Proposed Standard). RFC. Obsoleted by RFCs 5246, 6066, updated by RFC 5746. April 2006. URL: https://www.rfc-editor.org/rfc/rfc4366.txt (page 134).

[257] Lin-Shung Huang, Shrikant Adhikarla, Dan Boneh and Collin Jackson. "An Experimental Study of TLS Forward Secrecy Deployments". In: *Internet Computing, IEEE* 18.6 (November 2014), pp. 43–51. DOI: 10.1109/MIC.2014.86 (page 134).

[258] Brad Miller, Ling Huang, Anthony D. Joseph and J.D. Tygar. "I Know Why You Went to the Clinic: Risks and Realization of HTTPS Traffic Analysis". In: *Privacy Enhancing Technologies*. Vol. 8555. Lecture Notes in Computer Science. Springer International Publishing, 2014, pp. 143–163 (page 134).

[259] Robert Koch and Gabi Dreo Rodosek. "Command Evaluation in Encrypted Remote Sessions". In: *Proceedings of the 2010 Fourth International Conference on Network and System Security*. NSS '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 299–305. ISBN: 978-0-7695-4159-4 (pages 134, 142, 143).

[260] Laurens Hellemons, Luuk Hendriks, Rick Hofstede, Anna Sperotto, Ramin Sadre and Aiko Pras. "SSHCure: A Flow-Based SSH Intrusion Detection System". In: *Dependable Networks and Services*. Vol. 7279. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, pp. 86–97 (page 134).

[261] ipoque GmbH. *PACE 2.0*. URL: http://www.ipoque.com/en/products/pace (Accessed on 25th November 2014) (page 136).

[262] ClearFoundation. *l7-filter*. October 2013. URL: http://l7-filter.clearfoundation.com/ (Accessed on 26th November 2014) (page 137).

[263] Ethem Alpaydin. *Introduction to Machine Learning*. 2nd. The MIT Press, 2010. ISBN: 978-0262012430 (page 138).

[264] Guang-Lu Sun, Yibo Xue, Yingfei Dong, Dongsheng Wang and Chenglong Li. "An Novel Hybrid Method for Effectively Classifying Encrypted Traffic". In: *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*. Miami, Florida, USA, December 2010, pp. 1–5 (pages 138, 143, 144).

[265] Yohei Okada, Shingo Ata, Nobuyuki Nakamura, Yoshihiro Nakahira and Ikuo Oka. "Application Identification from Encrypted Traffic Based on Characteristic Changes by Encryption". In: *Communications Quality and Reliability (CQR), 2011 IEEE International Workshop Technical Committee on*. Naples, Florida, USA, May 2011, pp. 1–6 (pages 138, 143).

[266] Daniel J. Arndt and A. Nur Zincir-Heywood. "A Comparison of Three Machine Learning Techniques for Encrypted Network Traffic Analysis". In: *Computational Intelligence for Security and Defense Applications (CISDA), 2011 IEEE Symposium on*. Paris, France, April 2011, pp. 107–114 (pages 139, 143, 144).

[267] Riyad Alshammari, Peter Lichodzijewski, I. Malcolm Heywood and A. Nur Zincir-Heywood. "Classifying SSH Encrypted Traffic with Minimum Packet Header Features Using Genetic Programming". In: *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*. GECCO '09. Montreal, Québec, Canada: ACM, 2009, pp. 2539–2546. ISBN: 978-1-60558-505-5 (pages 139, 142–144).

[268] Riyad Alshammari and A. Nur Zincir-Heywood. "A Flow Based Approach for SSH Traffic Detection". In: *Systems, Man and Cybernetics, 2007. ISIC. IEEE International Conference on*. Montreal, Quebec, Canada, October 2007, pp. 296–301 (pages 139, 143, 144).

[269] Riyad Alshammari and A. Nur Zincir-Heywood. "A Preliminary Performance Comparison of Two Feature Sets for Encrypted Traffic Classification". English. In: *Proceedings of the International Workshop on Computational Intelligence in Security for Information Systems CISIS'08*. Vol. 53. Advances in Soft Computing. Genoa, Italy: Springer Berlin Heidelberg, 2009, pp. 203–210. ISBN: 978-3-540-88180-3 (pages 139, 142–144).

[270] Riyad Alshammari and A. Nur Zincir-Heywood. "Machine learning based encrypted traffic classification: Identifying SSH and Skype". In: *Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on*. Ottawa, Canada, July 2009, pp. 1–8 (pages 139, 143, 144).

[271] Riyad Alshammari and A. Nur Zincir-Heywood. "An Investigation on the Identification of VoIP traffic: Case study on Gtalk and Skype". In: *Network and Service Management (CNSM), 2010 International Conference on*. Niagara Falls, Canada, October 2010, pp. 310–313 (pages 139, 143, 144).

[272] Riyad Alshammari and A. Nur Zincir-Heywood. "Can encrypted traffic be identified without port numbers, IP addresses and payload inspection?" In: *Computer Networks* 55.6 (2011), pp. 1326–1350. ISSN: 1389-1286. DOI: http://dx.doi.org/10.1016/j.comnet.2010.12.002 (pages 139, 142–144).

[273] Yuichi Kumano, Shingo Ata, Nobuyuki Nakamura, Yoshihiro Nakahira and Ikuo Oka. "Towards real-time processing for application identification of encrypted traffic". In: *Computing, Networking and Communications (ICNC), 2014 International Conference on*. Honolulu, Hawaii, USA, February 2014, pp. 136–140 (pages 139, 143, 144).

[274] Laurent Bernaille and Renata Teixeira. "Early Recognition of Encrypted Applications". In: *Proceedings of the 8th International Conference on Passive and Active Network Measurement*. PAM'07. Louvain-la-Neuve, Belgium: Springer-Verlag, 2007, pp. 165–175. ISBN: 978-3-540-71616-7 (pages 139, 143, 144).

[275] Gianluca Maiolini, Andrea Baiocchi, Alfonso Iacovazzi and Antonello Rizzi. "Real Time Identification of SSH Encrypted Application Flows by Using Cluster Analysis Techniques". English. In: *NETWORKING 2009*. Vol. 5550. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009, pp. 182–194. ISBN: 978-3-642-01398-0 (pages 139, 143, 144).

[276] Carlos Bacquet, A. Nur Zincir-Heywood and Malcolm I. Heywood. "An Investigation of Multi-objective Genetic Algorithms for Encrypted Traffic Identification". English. In: *Computational Intelligence in Security for Information Systems*. Vol. 63. Advances in Intelligent and Soft Computing. Springer Berlin Heidelberg, 2009, pp. 93–100. ISBN: 978-3-642-04090-0 (pages 140, 143, 144).

[277] Carlos Bacquet, A. Nur Zincir-Heywood and Malcolm I. Heywood. "Genetic Optimization and Hierarchical Clustering Applied to Encrypted Traffic Identification". In: *Computational Intelligence in Cyber Security (CICS), 2011 IEEE Symposium on*. Paris, France, April 2011, pp. 194–201 (pages 140, 143, 144).

[278] Roni Bar-Yanai, Michael Langberg, David Peleg and Liam Roditty. "Realtime Classification for Encrypted Traffic". English. In: *Experimental Algorithms*. Vol. 6049. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, pp. 373–385. ISBN: 978-3-642-13192-9 (pages 140, 143, 144).

[279] Meng Zhang, Hongli Zhang, Bo Zhang and Gang Lu. "Encrypted Traffic Classification Based on an Improved Clustering Algorithm". English. In: *Trustworthy Computing and Services*. Vol. 320. Communications in Computer and Information Science. Springer Berlin Heidelberg, 2013, pp. 124–131. ISBN: 978-3-642-35794-7 (pages 140, 143, 144).

[280] Ye Du and Ruhui Zhang. "Design of a Method for Encrypted P2P Traffic Identification Using K-means Algorithm". English. In: *Telecommunication Systems* 53.1 (2013), pp. 163–168. ISSN: 1018-4864. DOI: 10.1007/s11235-013-9690-5 (pages 140, 143).

[281] Annie De Montigny-Leboeuf. "Flow attributes for use in traffic characterization". In: *Communications Research Centre Canada, Tech. Rep* (2005) (pages 140, 143).

[282] Yipeng Wang, Zhibin Zhang, Li Guo and Shuhao Li. "Using Entropy to Classify Traffic More Deeply". In: *Networking, Architecture and Storage (NAS), 2011 6th IEEE International Conference on*. Dalian, Liaoning, China, July 2011, pp. 45–52 (pages 140, 143, 144).

[283] Maciej Korczynski and Andrzej Duda. "Classifying Service Flows in the Encrypted Skype Traffic". In: *Communications (ICC), 2012 IEEE International Conference on*. Ottawa, Canada, June 2012, pp. 1064–1068 (pages 141, 143, 144).

[284] Payam Vahdani Amoli and Timo Hamalainen. "A Real Time Unsupervised NIDS for Detecting Unknown and Encrypted Network Attacks in High Speed Network". In: *Measurements and Networking Proceedings (M N), 2013 IEEE International Workshop on*. Naples, Italy, October 2013, pp. 149–154 (pages 141, 143).

[285] Maciej Korczynski and Andrzej Duda. "Markov Chain Fingerprinting to Classify Encrypted Traffic". In: *INFOCOM, 2014 Proceedings IEEE*. April 2014, pp. 781–789 (pages 141, 143, 144).

[286] Thomas Karagiannis, Konstantina Papagiannaki and Michalis Faloutsos. "BLINC: Multilevel Traffic Classification in the Dark". In: *Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. SIGCOMM '05. Philadelphia, Pennsylvania, USA: ACM, 2005, pp. 229–240. ISBN: 1-59593-009-4 (pages 141, 143, 144).

[287] Charles V. Wright, Fabian Monrose and Gerald M. Masson. "On Inferring Application Protocol Behaviors in Encrypted Network Traffic". In: *J. Mach. Learn. Res.* 7 (December 2006), pp. 2745–2769. ISSN: 1532-4435 (pages 142, 143).

[288] Amir R. Khakpour and Alex X. Liu. "An Information-Theoretical Approach to High-Speed Flow Nature Identification". In: *Networking, IEEE/ACM Transactions on* 21.4 (August 2013), pp. 1076–1089. ISSN: 1063-6692. DOI: 10.1109/TNET.2012.2219591 (pages 142–144).

[289] CAIDA. *Cyber-security Research Ethics Dialog & Strategy Workshop*. May 2013. URL: http://www.caida.org/workshops/creds/1305/ (Accessed on 22nd February 2018) (page 145).

[290] IETF. *IETF 88 Proceedings, Technical Plenary*. 2014. URL: http://www.ietf.org/proceedings/88/technical-plenary.html (Accessed on 3rd December 2014) (page 146).

[291] Yu Wang, Yang Xiang, Jun Zhang, Wanlei Zhou and Bailin Xie. "Internet traffic clustering with side information". In: *Journal of Computer and System Sciences* 80.5 (2014). Special Issue on Dependable and Secure Computing The 9th {IEEE} International Conference on Dependable, Autonomic and Secure Computing, pp. 1021–1036. ISSN: 0022-0000. DOI: http://dx.doi.org/10.1016/j.jcss.2014.02.008 (page 148).

[292] Harsha V. Madhyastha and Balachander Krishnamurthy. "A Generic Language for Application-specific Flow Sampling". In: *SIGCOMM Comput. Commun. Rev.* 38.2 (March 2008), pp. 5–16. ISSN: 0146-4833. DOI: 10.1145/1355734.1355736 (page 148).

[293] Myungjin Lee, Mohammad Hajjat, Ramana Rao Kompella and Sanjay G. Rao. "A Flow Measurement Architecture to Preserve Application Structure". In: *Comput. Netw.* 77.C (February 2015), pp. 181–195. ISSN: 1389-1286. DOI: 10.1016/j.comnet.2014.11.005 (page 148).

[294] Yan Hu, Dah-Ming Chiu and John C. S. Lui. "Entropy Based Adaptive Flow Aggregation". In: *IEEE/ACM Trans. Netw.* 17.3 (June 2009), pp. 698–711. ISSN: 1063-6692. DOI: 10.1109/TNET.2008.2002560 (page 148).

[295] Lautaro Dolberg, Jérôme François and Thomas Engel. "Efficient Multidimensional Aggregation for Large Scale Monitoring". In: *Proceedings of the 26th International Conference on Large Installation System Administration: Strategies, Tools, and Techniques*. lisa'12. San Diego, CA: USENIX Association, 2012, pp. 163–180 (page 149).

Appendix A

List of Authored Publications

A.1 Impacted Journals

1. Petr Velan, Milan Čermák, Pavel Čeleda, and Martin Drašar. "A Survey of Methods for Encrypted Traffic Classification and Analysis". In: *International Journal of Network Management* 25.5 (2015), pp. 355–374. DOI: 10.1002/nem.1901
   - Impact Factor: **0.681**, Contribution: **50%**

A.2 Conference Proceedings

1. Petr Velan, Husák Martin, and Daniel Tovarňák. "Rapid Prototyping of Flow-Based Detection Methods Using Complex Event Processing". In: *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*. 2018, Accepted for publication, 3 pages
   - CORE Ranking: **B**, Demo paper, Contribution: **50%**

2. Petr Velan. "Improving Network Flow Definition: Formalization and Applicability". In: *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*. 2018, Accepted for publication, 5 pages
   - CORE Ranking: **B**, Contribution: **100%**

3. Luuk Hendriks, Petr Velan, Ricardo de O. Schmidt, Pieter-Tjerk de Boer, and Aiko Pras. "Threats and Surprises behind IPv6 Extension Headers". In: *2017 Network Traffic Measurement and Analysis Conference (TMA)*. IEEE Xplore Digital Library, June 2017, pp. 1–9
   - CORE Ranking: –, Contribution: **10%**

4. Luuk Hendriks, Petr Velan, Ricardo de O. Schmidt, Pieter-Tjerk de Boer, and Aiko Pras. "Flow-Based Detection of IPv6-specific Network Layer Attacks". In: *Security of Networks and Services in an All-Connected World: 11th IFIP WG 6.6 International Conference on Autonomous Infrastructure, Management, and Security, AIMS 2017, Zurich, Switzerland, July 10-13, 2017, Proceedings*. Cham: Springer International Publishing, 2017, pp. 137–142. ISBN: 978-3-319-60774-0
   - CORE Ranking: –, Contribution: **10%**

5. Petr Velan, Jana Medková, Tomáš Jirsík, and P. Čeleda. "Network Traffic Characterisation Using Flow-Based Statistics". In: *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*. April 2016, pp. 907–912
   - CORE Ranking: **B**, Contribution: **40%**

6. Petr Velan. "EventFlow: Network Flow Aggregation Based on User Actions". In: *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*. April 2016, pp. 767–

771
- CORE Ranking: **B**, Contribution: **100%**

7. Petr Velan and Viktor Puš. "High-Density Network Flow Monitoring". In: *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. May 2015, pp. 996–1001
   - CORE Ranking: **A**, Contribution: **70%**

8. Viktor Puš, Petr Velan, Lukáš Kekely, Jan Kořenek, and Pavel Minařík. "Hardware Accelerated Flow Measurement of 100 Gb Ethernet". eng. In: *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. Ottawa, Canada: IEEE Xplore Digital Library, May 2015, pp. 1147–1148
   - CORE Ranking: **A**, Demo paper, Contribution: **50%**

9. Martin Husák, Petr Velan, and Jan Vykopal. "Security Monitoring of HTTP Traffic Using Extended Flows". In: *2015 10th International Conference on Availability, Reliability and Security*. August 2015, pp. 258–265
   - CORE Ranking: –, FARES Workshop, Contribution: **20%**

10. Petr Velan and Pavel Čeleda. "Next Generation Application-Aware Flow Monitoring". English. In: *Monitoring and Securing Virtualized Networks and Services*. Vol. 8508. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2014, pp. 173–178. ISBN: 978-3-662-43861-9
    - CORE Ranking: –, Contribution: **90%**

11. Petr Velan, Tomáš Jirsík, and Pavel Čeleda. "Design and Evaluation of HTTP Protocol Parsers for IPFIX Measurement". In: *Advances in Communication Networking*. Vol. 8115. Heidelberg: Springer Berlin Heidelberg, 2013, pp. 136–147. ISBN: 978-3-642-40551-8
    - CORE Ranking: –, Contribution: **50%**

12. Petr Velan. "Practical Experience with IPFIX Flow Collectors". In: *IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*. Ghent, Belgium: IEEE Xplore Digital Library, May 2013, pp. 1021–1026. ISBN: 978-1-4673-5229-1
    - CORE Ranking: **A**, Contribution: **100%**

13. Martin Elich, Petr Velan, Tomáš Jirsík, and Pavel Čeleda. "An Investigation Into Teredo and 6to4 Transition Mechanisms: Traffic Analysis". In: *IEEE 38th Conference on Local Computer Networks Workshops (LCN Workshops)*. Sydney, Australia: IEEE Xplore Digital Library, October 2013, pp. 1046–1052. ISBN: 978-1-4799-0540-9
    - CORE Ranking: –, WNM Workshop, Contribution: **25%**

14. Pavel Čeleda, Petr Velan, Martin Rábek, Rick Hofstede, and Aiko Pras. "Large-Scale Geolocation for NetFlow". In: *IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*. Ghent, Belgium: IEEE Xplore Digital Library, May 2013, pp. 1015–1020. ISBN: 978-1-4673-5229-1
    - CORE Ranking: **A**, Contribution: **20%**

15. Petr Velan and Radek Krejčí. "Flow Information Storage Assessment Using IPFIXcol". In: *Dependable Networks and Services*. Vol. 7279. Lecture Notes in Computer Science. Heidelberg: Springer Berlin Heidelberg, 2012, pp. 155–158. ISBN: 978-3-642-30632-7
    - CORE Ranking: –, Contribution: **70%**