Chair of Network Architectures and Services
School of Computation, Information, and Technology
Technical University of Munich

TECHNICAL UNIVERSITY OF MUNICH

SCHOOL OF COMPUTATION, INFORMATION, AND TECHNOLOGY

INFORMATICS

BACHELOR'S THESIS IN INFORMATICS

Packet Selection using Concepts from IPFIX and PSAMP

Tristan Döring

Technical University of Munich

School of Computation, Information, and Technology

Informatics

Bachelor's Thesis in Informatics

Packet Selection using Concepts from IPFIX and PSAMP

Packet Selection unter Verwendung von Konzepten aus IPFIX und PSAMP

Author:	Tristan Döring
Supervisor:	Prof. Dr.-Ing. Georg Carle
Advisor:	Kilian Holzinger, Henning Stubbe
Date:	May 15, 2023

I confirm that this Bachelor's Thesis is my own work and I have documented all sources and material used.

Garching, May 15, 2023

Location, Date

Signature

Abstract

In this thesis, we propose a flexible and versatile packet selection architecture based on the IPFIX/PSAMP standards, aiming to facilitate a wide range of network functions while maintaining compatibility with existing Internet standards.

Currently, there is no widely adopted standard for packet selection mechanisms in network functions. The IPFIX/PSAMP standards provide a versatile and standardized foundation for packet selection purposes, but existing research utilizing IPFIX/PSAMP concepts for network function development is inhomogeneous, often involves redundant efforts, and typically revolves around the implementation of a specific network function.

To address these issues, this thesis identifies the useful packet selection concepts and mechanisms within IPFIX/PSAMP and derives a comprehensive architecture draft from them with the goal of simplifying network function implementations. The resulting architecture is applicable to a broad spectrum of network functions, is designed for low-latency processing, and supports distributed network functions. It enables packet sampling and packet filtering while offering capabilities to filter header fields of various protocols used in network packets.

We verify this flexible packet selection architecture by performing a case study on Network Health Monitoring and a Network Coding network function. Furthermore, we develop two architecture implementations using Python and the open-source software Vermont. Both are tested to validate the architecture's functionality and identify possible extensions to Vermont required to support the new architecture fully.

Contents

1 Introduction **1**
1.1 Outline 2

2 Background **5**
2.1 Introduction to IPFIX/PSAMP 5
2.1.1 Architectural Similarities 5
2.1.2 General Simplification of the architecture 6
2.1.3 The Key Difference between IPFIX and PSAMP: the Metering Process 6
2.1.4 IPFIX/PSAMP Information Elements (IEs) 8
2.1.5 IPFIX/PSAMP Protocol 8
2.2 Network Function Use Cases 9

3 Related Work **11**

4 Problem Analysis **15**
4.1 Problem Statement 15
4.2 Background and Context 15
4.2.1 Available packet selection frameworks 16
4.3 Case study 17
4.3.1 Network Health Monitoring 17
4.3.2 Network Coding: Tetrys 19
4.3.3 Packet Selection Architecture Requirements 19
4.3.4 Comparison of available solutions 20
4.4 Methodological Approach for the Development of an IPFIX/PSAMP-based Flexible Packet Selection Architecture 22

5 Flexible Packet Selection Architecture **23**
5.1 Identification of relevant IPFIX/PSAMP components 23

5.1.1 PSAMP as the architecture basis . . . 23
5.1.2 Meeting the Architecture Requirements using PSAMP . . . 24
5.2 Placement of the Network Function within the Architecture . . . 27
5.3 Architecture Description . . . 28

6 Design of the Network Functions **31**
6.1 Information Flow . . . 31
6.2 Network Health Monitoring . . . 32
6.2.1 Advantages of the Packet Selection Architecture for Network Health Monitoring . . . 32
6.2.2 Flow Metrics . . . 33
6.3 Network Coding: Tetrys . . . 35
6.3.1 The Tetrys Architecture Concept . . . 35

7 Implementation **39**
7.1 Analysis of available Implementations . . . 39
7.1.1 PSAMP Collector Implementation . . . 39
7.1.2 PSAMP Device Implementation . . . 40
7.2 Python Prototype Implementations of PSAMP Device and Collector . . . 44
7.2.1 Overview . . . 44
7.2.2 Development Process and Implementation . . . 44
7.3 Conclusion . . . 50

8 Evaluation **51**
8.1 Test Methodology . . . 51
8.1.1 Test Scenario - Dual Camera . . . 53
8.1.2 Reproducibility of Test Results . . . 53
8.2 Network Health Monitoring Tests . . . 54
8.2.1 Packet Rate . . . 54
8.2.2 Packet Interarrival Times: Python Prototype vs. Vermont . . . 61
8.2.3 Throughput . . . 64
8.2.4 Performance Comparison: Python Prototype vs. Vermont . . . 66
8.3 Tetrys Architecture Concept Test Scenario . . . 69
8.3.1 Test Scenario Definition . . . 69
8.3.2 Tetrys Architecture Test Behaviour . . . 70
8.4 Flexibility Evaluation of the Architecture . . . 71
8.4.1 Flexibility Limitations . . . 72
8.5 Summary . . . 73

9 Conclusion **75**
9.1 Summary of results 75
9.2 Future Work 76

A Appendix **79**
A.1 List of Acronyms 79

Bibliography **81**

List of Figures

2.1 IPFIX/PSAMP Architecture high-level view as described in RFC 5470 [8] 6
2.2 Comparison of IPFIX and PSAMP Metering Processes as described in RFC5476[7] 7

5.1 IPFIX/PSAMP Metering Process composition as described in RFC 6728 [25] 24
5.2 Placement options for the network function within the IPFIX/PSAMP architecture, position two achieves low result availability delay and facilitates distributed network functions 28
5.3 Final packet selection architecture composition 29

6.1 Information flow inside the PSAMP Collector of the packet selection architecture 32
6.2 Tetrys Architecture Concept 36

8.1 Python prototype: Test setup on the test bed environment 52
8.2 Vermont prototype: Test setup on the test bed environment 52
8.3 Python prototype: packet rate measurement (Report Interpretation-based, export interval: 0.2 s, no Packet Report export) 56
8.4 Python prototype: packet rate measurement (Report Interpretation-based, export interval: 0.1 s, no Packet Report export), the decrease of the export interval from 0.2 s to 0.1 s reduces the accuracy 57
8.5 Python prototype: packet rate measurement comparison (Report Interpretation-based, export interval: 0.2 s), the inclusion of Packet Report export reduces the accuracy 58
8.6 Python prototype: packet rate measurement (Packet Report-based, granularity: 0.2 s), the measurement is accurate 60
8.7 Python prototype: packet interarrival times from Packet Reports, the measurement is accurate 62

8.8 Vermont prototype: packet interarrival time measurement (Packet Report-based), the accuracy is limited to milliseconds by the utilized Information Element (Listing 8.5) 64
8.9 Python prototype: data throughput measurement (Packet Report-based, granularity: 0.2 s), the measurement is accurate 65
8.10 Python prototype: packet rate measurement (performance test scenario: **TS 1000**, Packet Report loss: 83.45 %), an example of critical Packet Report loss 68
8.11 Tetrys Architecture Concept: packet loss scenario, the packet loss increases linearly over time, the Tetrys Architecture is expected to react by increasing the coding redundancy proportionally to the packet loss . 70

List of Tables

4.1 Comparison of packet selection frameworks (packet selection architecture requirement analysis), IPFIX/PSAMP stand out as the most viable solutions 21

6.1 Enterprise-specific Information Elements for the Tetrys Architecture Concept 36
6.2 Existing Information Elements for the Tetrys Architecture Concept from the IANA IPFIX registry [9] 36

7.1 Comparison of PSAMP Device implementations (PSAMP Device requirement analysis), only Vermont fulfills all crucial requirements **R1-R3** . . 41

8.1 Test scenario: Dual Camera, simulation of network traffic from a computer vision application 53
8.2 Performance test scenarios: Dual Camera + background traffic, the inclusion of background traffic enables performance benchmarking of the PSAMP Device implementations 66
8.3 PSAMP Device Performance Test Scenarios 66
8.4 PSAMP Device Performance Test Results, Vermont is perfectly reliable throughout the tests, the Python prototype experiences varying degrees of Packet Report loss 67

Chapter 1

Introduction

The growing complexity and variety of network functions demand effective and efficient ways of handling and processing network traffic. One common requirement for these network functions is the ability to apply them to packets sharing specific characteristics selectively. Currently, many network functions rely on individual implementations of packet selection mechanisms to accomplish this functionality. However, this may result in varying levels of effectiveness. Therefore, exploring a general packet selection architecture that is flexible and suitable for a wide range of network functions is essential. Such an architecture holds potential benefits for network functions, including faster development speed, improved quality, and increased interoperability.

The IPFIX (IP Flow Information Export) and PSAMP (Packet Sampling) internet standards offer promising foundations for such an architecture, including mechanisms and data models for packet selection and handling selection results. This thesis uses these standards to conceptualize a new, flexible packet selection architecture.

The primary goal of this thesis is to identify IPFIX/PSAMP architectural components and data models that can be reused to develop a flexible packet selection architecture. The potential of this new architecture will be analyzed and tested for two modern network functions: Network Health Monitoring and the Network Coding protocol Tetrys.

The scientific relevance of this thesis lies in the novel application of IPFIX/PSAMP, focusing on general packet selection, which leads to new network architectures, e.g. incorporating both IPFIX/PSAMP and Tetrys. It is argued that a flexible packet selection architecture, based on components from IPFIX/PSAMP, offers development and operational advantages for a wide range of network functions, as the standards are intended for use in a variety of application domains [1]. The additional generalization is ex-

pected to improve compatibility with other network functions not previously considered in conjunction with IPFIX/PSAMP.

To demonstrate and validate the functionality of the proposed architecture, it is implemented utilizing the open-source software Vermont [2] and a pair of Python prototypes. These tools will be used to evaluate the architecture's behavior and performance in various test scenarios. The proposed architecture is expected to promote further development and integration of IPFIX/PSAMP in network applications. Furthermore, based on the findings of this thesis, an extension to Vermont that includes the capabilities of the new packet selection architecture is encouraged.

1.1 Outline

The thesis continues with the background Chapter 2, where the essential IPFIX and PSAMP architecture principles and mechanisms are explained. Additionally, the relationship between IPFIX and PSAMP is clarified, laying the foundation for the following chapters.

Chapter 3 provides an overview of existing IPFIX/PSAMP research, focusing on previous applications of IPFIX and PSAMP demonstrating its adaptability for various network functions.

Next, Chapter 4 assesses several candidates as the basis for defining a flexible packet selection architecture. Essential architecture requirements are outlined to determine the most suitable choice for the proposed architecture.

Chapter 5 conceptualizes the flexible packet selection architecture based on IPFIX/PSAMP. This architecture is designed to fulfill all the identified requirements, ensuring a robust and adaptable solution.

Chapter 6 discusses the architectural benefits and possibilities of the proposed packet selection architecture for Network Health Monitoring and Tetrys. This exploration demonstrates the real-world applicability and advantages of the new architecture.

Chapter 7 covers the realization of the proposed architecture, including the network functions. This chapter also includes an analysis of various open-source IPFIX/PSAMP implementations and leads to the development of Python prototypes.

In Chapter 8, the architecture implementations are applied to support a computer vision application, with various test scenarios assessing their performance and effectiveness. The chapter showcases the Network Health Monitoring and Tetrys use cases. Finally, the chapter concludes with an assessment of the architecture's flexibility.

In the final chapter (Chapter 9), the key findings of the thesis are summarized, and their relevance to other research publications is discussed. To conclude, the chapter presents possible directions for future research and identifies areas for further investigation in this domain.

Chapter 2

Background

This chapter provides the reader with the background information necessary to understand the following thesis.

2.1 Introduction to IPFIX/PSAMP

IPFIX and PSAMP are two interdependent standards. With the goal of a Flow-based monitoring architecture, IPFIX defines a comprehensive information model [3] and the IPFIX protocol [4]. PSAMP, a packet reporting framework, standardizes packet selection techniques [5] and outlines the structure of a PSAMP Device [6] and the PSAMP protocol [7] which is an extension of the IPFIX protocol.

2.1.1 Architectural Similarities

On a high architectural level, IPFIX and PSAMP work analogously because PSAMP is based on IPFIX while expanding on its concepts.

As shown in Figure 2.1 the architecture consists of two main components: the IPFIX/PSAMP Device and the IPFIX/PSAMP Collector [8]. A network packet is observed by the Observation Point within the IPFIX/PSAMP. After that, the packet is handed to the Metering Process which extracts information from the packets. The internal processes of the Metering Process vary from IPFIX to PSAMP and will be examined later. The information extracted from the packets is packed into IPFIX/PSAMP protocol messages that can be exported to the Collector. The Collector parses the protocol messages and stores the information.

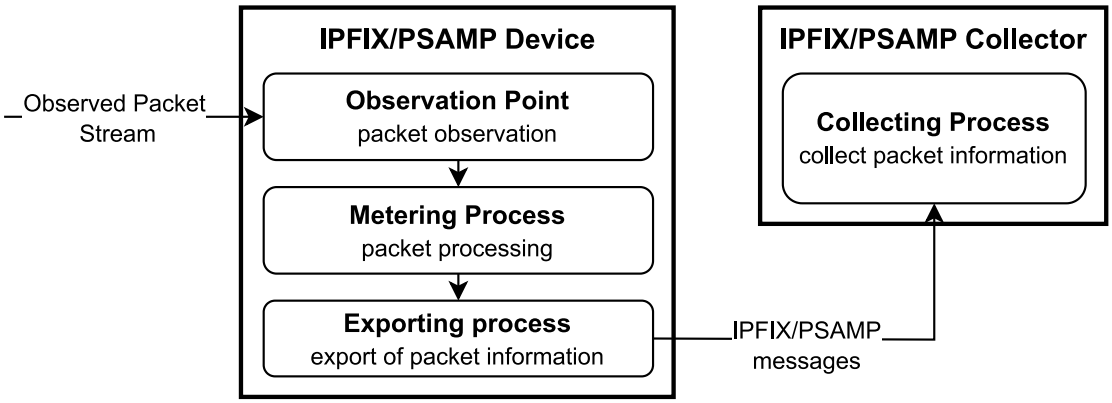

Figure 2.1: IPFIX/PSAMP Architecture high-level view as described in RFC 5470 [8]

2.1.2 General Simplification of the architecture

In order to simplify the concepts and explanations within this thesis, a general architecture simplification was applied. The following will discuss the differences between this simplified architecture and the original design.

According to the standard, an IPFIX/PSAMP Device can consist of multiple Observation Points, Metering Processes, and Exporting Processes. Furthermore, multiple IPFIX/PSAMP Devices can export to one or more Collecting Processes within one or more Collectors. For the sake of simplicity, if not otherwise mentioned, we will examine the base case of a single IPFIX/PSAMP Device configured with a single Observation Point, Metering Process, and Exporting Process and a Collector with one Collecting Process. Additionally, there is only one instance of an IPFIX/PSAMP Device and one Collector, although it is possible to have multiple of each as part of a more complex architecture. While simplified, this architecture is IPFIX/PSAMP conformant and is easily expanded for more complex architecture compositions.

2.1.3 The Key Difference between IPFIX and PSAMP: the Metering Process

The primary distinction between IPFIX and PSAMP lies in the fact that IPFIX focuses on the export of Flow Records, while PSAMP is concerned with the export of Packet Reports. Both of these processes rely on the inner workings of the Metering Process. The IPFIX standard describes an architecture for Flow-based IP traffic measurements, focusing on Packet Flows. A Flow consists of packets sharing common properties, known as Flow Keys, which characterize a single Flow. A typical example of a Flow Key is the IP 5-tuple, consisting of the IP source and destination address, source and destination port, and the transport protocol. In IPFIX, network packets and associated metadata

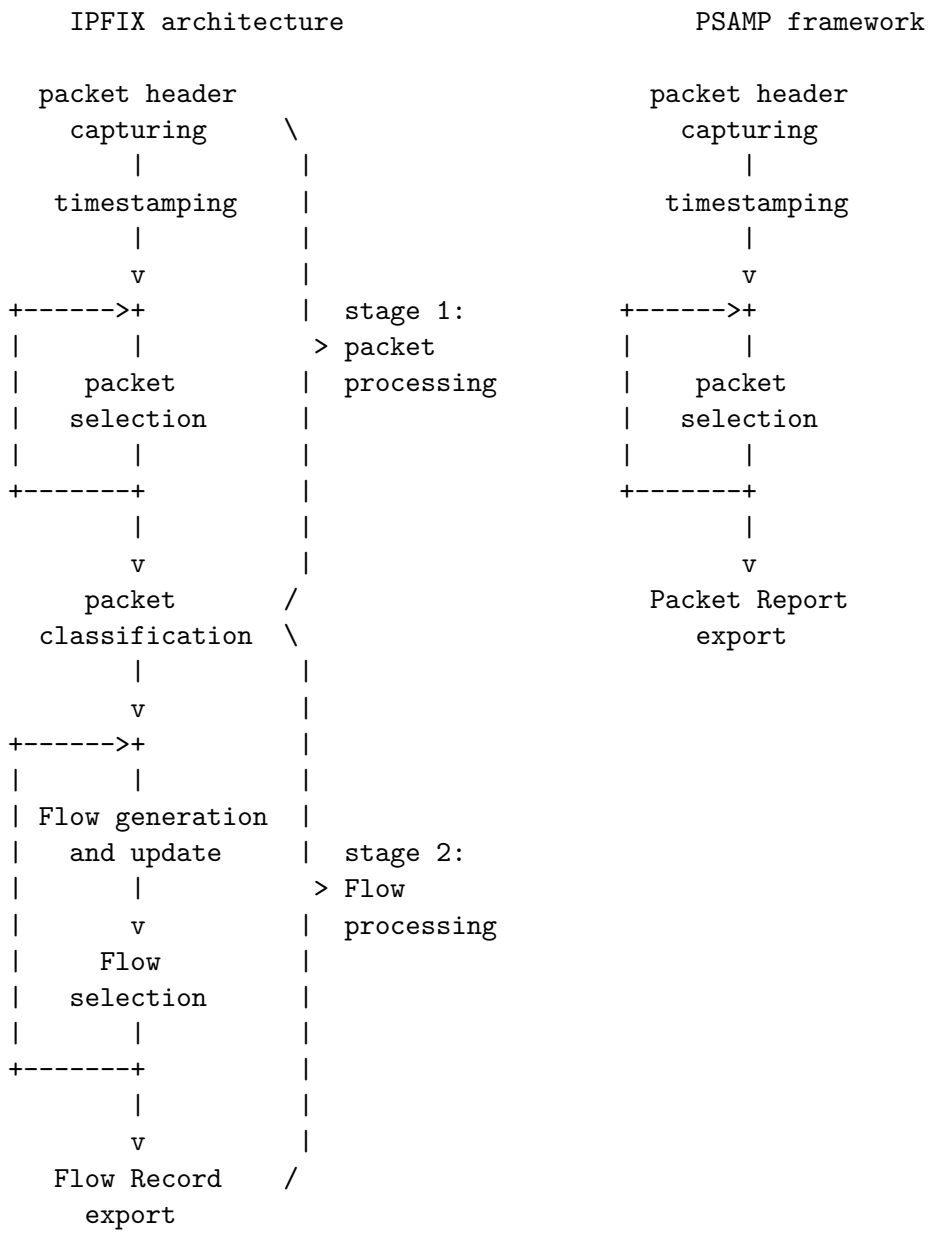

Figure 2.2: Comparison of IPFIX and PSAMP Metering Processes as described in RFC5476[7]

are aggregated into Flow Records, representing metrics of a Packet Flow [8]. PSAMP, on the other hand, is a framework for packet reporting [5] and defines packet selection techniques for this purpose. The selection outcome determines whether a packet and its metadata are reported on. PSAMP produces its reports on individually selected packets, called Packet Reports, without aggregating network packets into Packet Flows.

Comparing the inner structure of the IPFIX Metering Process in Figure 2.2 to the PSAMP Metering Process reveals a clear difference. The IPFIX Architecture defines two stages: packet processing and Flow processing. The first stage is functionally identical to the PSAMP Metering Process, consisting of packet header capturing, timestamping, and packet selection. The IPFIX Architecture utilizes the exact PSAMP packet selection mechanisms, but its use is optional for IPFIX [7]. The primary distinction between the two is the Flow processing stage, which is exclusive to IPFIX. As a result, the PSAMP Framework can be considered a subset of the IPFIX Architecture in terms of the Metering Process.

2.1.4 IPFIX/PSAMP Information Elements (IEs)

The IPFIX standard includes a formal description of its Information Elements. The IPFIX Information Elements registry is maintained by the IANA organization [9]. This registry defines a set of possible Information Elements to use with IPFX/PSAMP-compliant devices. Each of these Information Elements includes an `elementId`, a defined datatype, and a clear description of its meaning. This ensures consistency in the interpretation of data at each component. Some of these Information Elements are meant for exclusive use with PSAMP. Furthermore, the standards allow the definition of custom enterprise-specific Information Elements.

2.1.5 IPFIX/PSAMP Protocol

The IPFIX protocol aims to export Flow-related information from the IPFIX Device to the IPFIX Collector. The IPFIX protocol outlines three distinct types of information entities, known as Records, which are transmitted within an IPFIX message.

- Template Records define the structure of Data Records, specifying which Information Element fields are included in a Data Record.
- Options Template Records are Template Records that additionally determine Information Elements used to scope the Data Record. They can be used to define Data Records that are not directly related to packets or Flows.
- Data Records contain multiple values for Information Elements, with the structure being defined in a Template Record or Options Template Record. Data Records carry the actual information. In the context of IPFIX, Data Records typically represent Flow Records.

The PSAMP protocol is based on the IPFIX protocol and uses the same protocol messages and structure. The goal of exporting information from the PSAMP Device to the

PSAMP Collector is identical. PSAMP specifies two distinct information entities, both IPFIX protocol Data Records instances.

- A PSAMP Packet Report is considered a special case of an IPFIX Flow Record that reports only on a single packet. The structure and interpretation are defined by a Template Record.
- A PSAMP Report Interpretation is a Data Record whose structure is defined by an Options Template Record. Report Interpretations typically represent statistical or configuration information about the PSAMP Device.

PSAMP makes use of exclusive Information Elements specific to its framework. These include properties related to PSAMP concepts, such as those associated with packet selection [10].

2.2 Network Function Use Cases

In this thesis, the proposed packet selection architecture demonstrates flexibility by analysis of two specific use cases: Network Health Monitoring and the Network Coding protocol Tetrys.

A Network Health Monitoring network function is a system that observes, measures and analyzes the performance and status of a network. It gathers metrics such as packet rates, throughput, and packet interarrival times to assess the overall network health, enabling network administrators to detect and troubleshoot issues, optimize performance, and maintain network stability.

A Network Coding Protocol, such as Tetrys [11], is a system that enhances the reliability and efficiency of data transmission in a network by applying coding techniques to the original data packets. Tetrys, in particular, operates by introducing redundancy to improve the resilience of the transmitted data against packet loss.

Chapter 3

Related Work

A significant amount of research has been conducted on the application of IPFIX/PSAMP to accomplish various network functions. These studies are typically focused on specific use cases, highlighting the adaptability of IPFIX/PSAMP for different network functions.

One common application for IPFIX/PSAMP is Flow-based network monitoring, as demonstrated in [12]. The papers emphasize the importance of the distributed nature of IPFIX/PSAMP, which enables efficient traffic monitoring at various Observation Points using a central network analyzer. IPFIX Flow aggregation is applied to wireless network protocols in order to determine fine-grained Quality of Service (QoS) metrics of the heterogeneous network traffic. The adaption of IPFIX/PSAMP to uncommon network protocols is achievable due to the flexibility in defining Packet Flows within IPFIX.

When comparing our flexible packet selection architecture to existing IPFIX/PSAMP applications in literature, the rarity of PSAMP Packet Report usage is apparent, as most IPFIX/PSAMP applications focus on Flow-based monitoring, including Flow-aggregation and Flow Records.

Munz and Carle present a notable exception in [13], outlining an architectural description for a distributed network analysis function. Their architecture follows the same fundamental principles as ours, placing the network function (in their case, a network analyzer) at the Collector, just as envisioned in our packet selection architecture. Their approach to distributed network analysis is fully compatible with our architecture, making their paper an excellent demonstration of a complex network function with practical use that can be implemented with our architecture. In [14], Kaneko et al. describe a precise network monitoring system designed to monitor high-quality video

live streaming. The paper combines packet selection with data extraction from packets, both mechanisms introduced in our architecture, to enable detailed network monitoring based on PSAMP. Consequently, this network function can also be implemented following our architecture. In their research, Kobayashi et al. [15] employ a combination of PSAMP packet filtering and sampling to measure IP Multicast Traffic. The approach includes the use of Report Interpretations to update the Multicast routing table state within the network monitor. Supplementary, enterprise-specific Information Elements are introduced to support the specific reporting requirements. The paper serves as an outstanding example of the extensive application of IPFIX and PSAMP principles to achieve targeted monitoring of complex network traffic while following the same ideas as our architecture. All three publications demonstrate the suitability of our architecture for precise per-packet network monitoring.

Furthermore, our architecture can also support untypical Flow-based network measurement architectures. The application presented in [16] incorporates a central Flow aggregation network function located at the PSAMP Collector. Packet Reports originating from two measurement nodes at different network locations are exported to the Collector to enable packet delay measurements between both network nodes. This architecture-compliant approach demonstrates the flexibility and adaptability of our architecture to accommodate unique measurement requirements.

Additionally, RFC 5472 [1] describes many applications for the IPFIX protocol. The various potential applications, including usage-based accounting, traffic profiling, traffic engineering, network security, QoS monitoring, inter-domain exchange of IPFIX data, and export of derived metrics, all benefit from the versatility of the IPFIX protocol utilized for exporting network traffic-related information. This versatility makes the IPFIX protocol an outstanding candidate for the export of packet selection results.

An example of adapting the IPFIX Architecture for a specific use case is presented in [17]. This study describes the application and slight modification of the IPFIX architecture to achieve real-time intrusion detection, showcasing the architecture's potential for customization to meet specific requirements.

The research papers mentioned earlier provide valuable examples and insights into utilizing IPFIX/PSAMP for packet selection, the transmission of network traffic data, and as an architectural base of network functions. In contrast to the previously mentioned works, this thesis aims to analyze IPFIX/PSAMP in a more general way, focusing on its potential as a protocol for packet selection result transmission and an architectural base for packet selection in network functions. To support this generalization, findings from various papers related to IPFIX/PSAMP, including the RFC standards, are

gathered and combined. The ultimate goal is not to provide another study focused on achieving specific network functions using IPFIX/PSAMP but rather to develop a flexible packet selection architecture that incorporates the most practical IPFIX/PSAMP concepts while remaining flexible toward various network functions.

Chapter 4

Problem Analysis

In this chapter, we will first provide an overview of existing packet selection solutions. These solutions, including IPFIX/PSAMP, will be compared to assess their viability for the desired flexible packet selection architecture. Finally, we will outline the further approach to developing the architecture.

4.1 Problem Statement

The modern Internet is a heterogeneous multipurpose network. That means a network stream contains packets of many different services. Many network functions should only be applied to specific service data streams, i.e. packets in a network stream that share specific properties. E.g. a network function should only be applied to packets of a video stream. All packets belonging to this video stream could e.g. be defined by their identical IP 5-tuple.

4.2 Background and Context

To implement such packet selection functionality, network functions require mechanisms to filter for relevant packets. For these filtering purposes, network functions typically utilize application-specific mechanisms. The selection logic and implementations of such a packet filter, however, vary from application to application. There is no common packet selection standard architecture used as a basis.

A packet selection standard would enable network functions to use common semantics and syntax for packet selection rules and therefore would make them interchangeable between them. Furthermore, implementations of packet selection mechanisms could be

more commonly reused across network function applications. This would result in a faster development speed, better quality due to development simplicity, and increased interoperability of network functions.

A packet selection standard is required to accomplish this, which would provide a consistent and interoperable basis for implementing packet selection across various network functions and applications. Unfortunately, there is currently no common packet selection standard or framework for that particular use case. However, a multitude of standards, frameworks, and tools that define packet selection mechanisms exist. The final goal of this thesis is to define a general packet selection architecture based on one of these available options or extensions of them. IPFIX/PSAMP is the primary candidate due to the reasons discussed in Chapter 3. However, thoroughly analyzing the most promising candidates is necessary to reveal their true potential.

4.2.1 Available packet selection frameworks

When it comes to packet selection, there are a variety of tools and frameworks available that provide different capabilities and features. In this section, we will be looking at four different frameworks: IPFIX/PSAMP, PF, POSIX sockets, and nftables. These frameworks have been selected due to their widespread use, diverse design, and relevance.

IPFIX/PSAMP is a pair of IETF standards that focus on Flow-based data collection and packet selection techniques, respectively. IPFIX (IP Flow Information Export) is a protocol for exporting Flow-based information from network devices to collectors [4]. It defines various data models, frameworks, and protocols to support collecting network traffic data. PSAMP (Packet Sampling) defines standard sampling, filtering techniques, and guidelines for their combination and configuration [6]. The standards are often used together to enable effective monitoring of network traffic.

PF, or Packet Filter, is a native packet filtering framework in BSD-based operating systems, offering robust and flexible capabilities for controlling network traffic based on various packet properties [18].

POSIX sockets provide a standard API for network programming [19]. As part of their functionality, POSIX sockets can listen on specific IP addresses and ports using various transport protocols such as TCP, UDP, and others. While they offer flexibility and portability, they do not inherently include built-in packet selection or filtering mechanisms.

Lastly, nftables is a modern packet filtering and classification framework introduced in the Linux kernel, designed to replace and enhance existing frameworks such as iptables,

ip6tables, arptables, and ebtables. nftables offers a more unified, flexible, and efficient approach to configuring packet filtering and network traffic processing rules [20].

We will analyze the key features regarding the packet selection capabilities of each of these frameworks. By comparing their capabilities and limitations, we aim to comprehensively understand their suitability for supporting network functions with packet selection functionality. A case study on two modern network functions, Network Health Monitoring and the Network Coding protocol Tetrys will be conducted to compare these diverse options comprehensively. This case study will help us understand the general packet selection architecture requirements.

4.3 Case study

In this section, we will present a case study examining the requirements imposed by two challenging network functions, Network Coding (Tetrys) and Network Health Monitoring, on packet selection mechanisms.

4.3.1 Network Health Monitoring

Network Health Monitoring involves measuring network statistics and traffic to ensure a network's health, which is determined by its ability to function reliably, timely, and efficiently. Packet selection mechanisms should possess several key capabilities to support Network Health Monitoring.

Firstly, they should be able to filter for various packet header fields across different networking layers. Some applications may assess packet behavior at the Ethernet, IP, or Transport Layer. Consequently, selection mechanisms should have access to all packet fields of different network layers to provide proper filtering capabilities. Additionally, other packet properties, such as packet size, may be of interest, as, e.g., the distribution of IP packet sizes can also impact network health [21].

To enable the use of a Network Health Monitor in diverse scenarios, it should be capable of filtering various kinds of network protocols, depending on the application. For instance, to be able to analyze newer trends in networking, such as the QUIC protocol [22]. Merely offering compatibility with many network protocols is insufficient for a general packet selection framework. To cope with this variety, it must also allow users to add new protocol capabilities by themselves to ensure true flexibility. Consequently, a packet selection framework should provide a well-defined mechanism for managing the inclusion of new filter properties.

Thirdly, the selection logic must provide sufficient complexity to filter for any combination of packet properties. Packet Flows, for example, often require filters that consider multiple properties simultaneously, such as the IP 5-tuple.

Furthermore, monitoring every packet is not always necessary or feasible due to performance limitations, e.g., in high bandwidth networks. Packet sampling techniques can reduce the Network Health Monitor's performance requirements by decreasing the number of packets that need to be processed. By doing so, a Network Health Monitor can approximate Packet Flow properties by analyzing just a fraction of packets. In contrast to packet filtering, packet sampling is a packet selection technique that does not always select based on packet properties but, e.g., selects random packets.

A technique to reduce the information input to Network Health Monitors and their complexity involves providing only the necessary information entities from a packet. This selective approach simplifies the monitoring process and minimizes the processing overhead, making it more efficient for Network Health Monitors to analyze the relevant data. For example, many Network Health Monitors are primarily interested in packet header fields rather than payload. Utilizing only the required information from the packet content would simplify Network Health Monitoring applications.

It is important to note that performance, such as packet throughput, is intentionally not included in these requirements. The focus is primarily on the capabilities and functional complexity, i.e., the range of metrics the Network Health Monitoring function can capture. However, low-latency processing and immediate availability of selection results are factors crucial for some Network Health Monitoring applications [23]. In this context, low latency is distinct from performance because it specifically refers to the minimal time delay between packet selection and the availability of results rather than the overall throughput or processing speed. Low-latency processing enables certain functions that rely on real-time decision-making, such as managing congested network segments.

Another requirement is the capacity to collect selection results from various locations within the network and centralize them for analysis by a single Network Health Monitoring function as demonstrated in [13]. Distributed observation of network packets can provide a more comprehensive perspective on network traffic and performance, reflecting real-world use cases. It enables monitoring complex topologies, for example, to localize faults or optimize load balancing.

The final requirement for packet selection mechanisms in Network Health Monitoring is being a standardized solution. Utilizing a standardized approach ensures interoperability between network functions and monitoring systems based on the same standard. This

promotes seamless integration and collaboration across various components of a network infrastructure. Standardized solutions also facilitate easier adoption, maintenance, and support, as they are generally well-documented and have established communities of users and developers.

4.3.2 Network Coding: Tetrys

Tetrys is a Network Coding protocol designed to enhance data transmission efficiency and reliability in communication networks. It achieves this by employing innovative coding techniques, which combine multiple data packets into a single coded packet [11].

Packet selection can benefit Tetrys in two ways. First, by enabling targeted selection of packets for encoding from network traffic, a packet selection architecture must be able to filter packets and forward whole packet captures to the Tetrys application. This targeted selection allows the Tetrys application to focus on ensuring Quality of Service (QoS) standards specifically for the filtered packets, which may represent important Packet Flows. This provision of full packet captures is a requirement to the selection architecture not yet addressed by the Network Health Monitoring use case.

Second, Tetrys can also benefit from the packet selection architecture being compatible with existing standards, e.g. for remote configuration purposes. In this way, Tetrys can be deployed alongside the packet selection architecture, while both are managed and coordinated by a control plane to achieve transmission reliability for selected packets. In general, this requirement facilitates the integration of the packet selection architecture into existing network architectures.

4.3.3 Packet Selection Architecture Requirements

The key requirements for an effective and flexible packet selection architecture for Network Health Monitoring can be summarized as follows:

R1: Layered Filtering - Ability to filter packets based on various header fields across different Layers of the networking stack and also other packet properties (e.g. packet size)

R2: Protocol Flexibility - Capability of being flexible in the inclusion of new or uncommon protocols

R3: Complex Packet Selection - Provision of sufficient complexity in the selection logic to filter for any combination of packet properties

R4: Packet Sampling - Incorporation of packet sampling techniques to reduce the performance requirements of network functions when handling high bandwidth networks

R5: Content Extraction - Extraction of only necessary packet content information entities to the network function for simplifying its operation and reducing the overall processing overhead.

R6: Low-Latency Processing - Immediate availability of selection results

R7: Distributed Collection - Capacity to collect selection results from various locations within the network for processing within a single network function

R8: Full Capture Provision - Inclusion of full packet captures for every packet in the selection result if necessary

R9: Standardized Solution - A standardized solution ensures interoperability, seamless integration, easy adoption, and collaboration across network components

A packet selection architecture can effectively support Network Health Monitoring, the Network Coding protocol Tetrys and other network functions by addressing these requirements.

4.3.4 Comparison of available solutions

In this section, we will compare the previously identified frameworks and standards involving packet selection. The goal is to assess which of these solutions is worth investigating in more detail.

PF can filter packets across Layers 2-4 of the networking stack [18]. Adding support for new protocols requires kernel modifications and recompilation, which is time-consuming. PF employs a flexible, rule-based approach for filtering [24], with immediate availability of filtered results as network packets. Although PF lacks packet sampling or information extraction features, it could be the filtering component for applications with these functionalities. However, it doesn't offer a framework for exporting or collecting packet selection results.

IPFIX/PSAMP enables filtering packet headers and meta-information, supporting packet header fields from Layers 2-4 [5]. It permits defining custom Information Elements for easy incorporation of new protocols [4]. Regarding the selection logic complexity, it enables the chaining of Selectors and combinations of packet filtering and sampling. Selection results can be defined to only export specific Information Elements. The latency of selection result availability depends on the system architecture,

Requirements	PF	IPFIX/PSAMP	POSIX sockets	nftables
R1: Layered Filtering	✓	✓	×	✓
R2: Protocol Flexibility	×	✓	×	×
R3: Complex Packet Selection	✓	✓	×	✓
R4: Packet Sampling	×	✓	×	×
R5: Content Extraction	×	✓	×	×
R6: Low-Latency Processing	✓	×	✓	✓
R7: Distributed Collection	×	✓	×	×
R8: Full Capture Provision	✓	(✓)	✓	✓
R9: Standardized Solution	✓	✓	✓	✓

Table 4.1: Comparison of packet selection frameworks (packet selection architecture requirement analysis), IPFIX/PSAMP stand out as the most viable solutions

as the framework separates packet observation and information collection. This principle enables the employment of distributed packet selectors. In typical deployments, selection result availability may be delayed by 165 seconds depending on configuration and implementation [17]. IPFIX is designed for Flow-based network monitoring using Flow Records without per-packet captures in its selection results. However, PSAMP introduces the export of Packet Reports that allows the inclusion of the entire packet content.

POSIX sockets offer limited packet filtering capabilities, allowing sockets to be configured to provide packets with a specific IP destination address, destination port, and transport protocol [19]. Beyond that, POSIX sockets do not offer any packet selection mechanisms. For more advanced packet selection, such mechanisms should be implemented using POSIX sockets as a base in a network application. POSIX provides such applications with low-latency access to network packets. The entire packet content is available to the application when using POSIX raw sockets.

The **nftables** packet classification framework enables filtering packet headers and meta information, such as packet length or network interface [20]. It supports packet header fields from Layers 2-4. Introducing new protocol headers to filtering lacks a simple mechanism. Packet selectors can be concatenated for complex selection rules, but neither sampling nor information extraction is supported. The selection result is the entire packet content. The framework does not define packet selection result distribution across the network. Nonetheless, selection results are immediately available.

The final results of the requirement analysis of the different packet selection solutions are summarized in Table 4.1.

In conclusion, it can be said that the IPFIX/PSAMP standards emerge as the most suitable choice in the comparison. That being said, IPFIX/PSAMP must be applied less conventionally to overcome the packet capture limitation. Furthermore, the high delay in selection result availability must be addressed and resolved, ideally within the constraints of the IPFIX/PSAMP standards.

4.4 Methodological Approach for the Development of an IPFIX/PSAMP-based Flexible Packet Selection Architecture

A solid methodological approach is required to define a packet selection architecture fulfilling the desired requirements.

First, the IPFIX/PSAMP standards must be applied to construct a draft for a flexible packet selection architecture that fulfills all the previously defined requirements while remaining true to the standard. We will then show how to design a Network Health Monitoring and Network Coding application compatible with the architecture. The architecture will be implemented and tested to demonstrate and validate the chosen approach to packet selection for network functions.

Chapter 5

Flexible Packet Selection Architecture

This chapter aims to construct a draft for a flexible packet selection architecture based on the IPFIX/PSAMP standards described in several RFCs released by the IETF. This architecture must fulfill the nine architecture requirements defined in the case study of Chapter 4. The correct components and mechanisms of IPFIX/PSAMP must be carefully identified and connected to achieve this. First, it will be described how we can achieve each requirement by using the IPFIX/PSAMP information models. At the same time, several components of IPFIX/PSAMP will be identified for use in the final architecture draft. Finally, a draft of the final architecture will be presented, including the relationship between components and the information flow.

5.1 Identification of relevant IPFIX/PSAMP components

5.1.1 PSAMP as the architecture basis

Both the IPFIX Architecture and the PSAMP Framework share components and functionalities. As elaborated in Chapter 2, the primary distinction between the two is the addition of Flow generation and processing in IPFIX. Flow aggregation results in a reduction of information, as packets belonging to the same Packet Flow are aggregated into a single Flow Record. Such a Flow Record contains information related to the Packet Flow as a whole but not to single packets. This information reduction is suitable for some edge case network functions only interested in Flow-related information. Still, it directly contradicts the objective of achieving broad compatibility with diverse network functions, e.g., Tetrys. Following the **Full Capture Provision (R8)** requirement, the proposed packet selection architecture aims to provide access to the

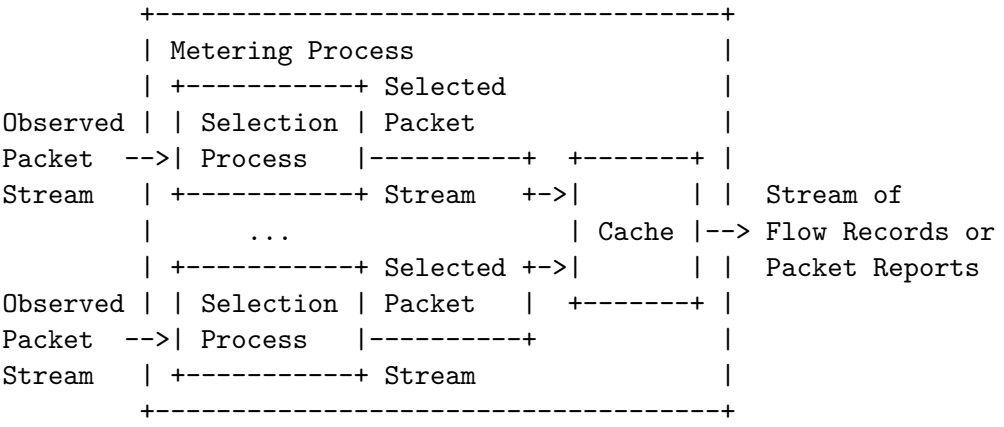

Figure 5.1: IPFIX/PSAMP Metering Process composition as described in RFC 6728 [25]

selection outcome of each packet. Consequently, the aggregation of Flows and Flow Records are outside the scope of this architecture. Given that PSAMP treats each packet individually, resulting in a Packet Report for every selected packet, the packet selection architecture will be based on the PSAMP Framework.

5.1.2 Meeting the Architecture Requirements using PSAMP

This section will discuss how the packet selection architecture requirements can be fulfilled using the PSAMP standards.

R3 - Complex Packet Selection, R4 - Packet Sampling

Understanding the selection logic within the Metering Process is necessary to analyze whether the packet selection logic available in PSAMP is sufficiently expressive. The composition of the Metering Process is depicted in Figure 5.1. Furthermore, an example of a typical selection scenario involving filtering for multiple packet properties will be examined at the end of this section to validate that the requirements are met using the proposed PSAMP packet selection mechanisms.

The packet selection functionality of PSAMP rests within the Selection Processes, components of the Metering Process. These Selection Processes consist of one or more Selectors. A Selector can either perform filtering or sampling. Consequently, a Selector will be configured for sampling or one of the two filtering techniques: Property Match Filtering or Hash-based Filtering [5].

Property Match Filtering selects a packet if specific fields or other packet properties of a packet match a specified value or fall within a defined value range. On the other hand, Hash-based Filtering utilizes a hash function on the packet content or a portion of it

and compares the outcome to a specified value or value range. Hash-Based Filtering can efficiently select packets sharing similar properties or enable quasi-random packet selection when applied to a large portion of the content.

A single Selector is called Primitive Selector. It is possible to combine Primitive Selectors in a chain-like structure. Each Selector of this ordered chain will be applied to the packet stream in the defined order. The output of the preceding Selector, i.e., the intermediate selection result, is input to the next Selector. With such Composite Selectors, it is possible to express reasonably complex packet selection schemes, including combinations of Hash-based Filtering, Property Match Filtering, and sampling. E.g., filtering for a Flow defined by an IP 5-tuple is easily achievable using a Composite Selector consisting of five Selectors, each configured to do Property Match Filtering for one of the IP 5-tuple's packet properties.

In addition to that, it is possible to use multiple Selection Processes within a Metering Process as displayed in Figure 5.1. These Selection Processes are applied in parallel to the same Observed Packet Stream, the input to the packet selection [6]. With this, multiple parallel Selection Processes results will result in a single stream of Packet Reports [25]. To maintain the relationship between a Packet Report and the Selection Process it originated from, it is necessary to include the Selection Process instance identifier `selectionSequenceId` into each Packet Report. Using these mechanics, it would be possible to classify an Observed Packet Stream by utilizing multiple Selection Processes, resulting in numerous Selected Packet Streams inside the Report Stream. Due to the parallelism of the Selection Processes, these Selected Packet Streams are not strictly partitions of the Observed Packet Stream, i.e., it is possible that they contain identical packets.

R1 - Layered Filtering
According to RFC 5475 [5], all so-called IPFIX Flow attributes can be used for Property Match Filtering. The current, complete list of IPFIX Flow attributes is included in the IANA registry for IPFIX Information Elements [9]. It includes support for various protocol headers of Layers 2-4 and derived packet properties such as the packet size.

R2 - Protocol Flexibility
The architecture must support the inclusion of new, currently unsupported protocols. To do so, the implementation needs to be able to parse new protocol headers, and the information model must incorporate new Information Elements. PSAMP facilitates such functional extensions through enterprise-specific Information Elements. These are designed to represent custom Information Elements that are used internally. Consequently,

new protocols can be included in a standard conform manner by adding enterprise-specific Information Elements for the necessary protocol header fields and properties.

R5 - Content Extraction, R8 - Full Capture Provision
Templates define the format of the selection results. Based on these, the Exporting Process, also located within the Metering Process, exports two different Data Records: Packet Reports and Report Interpretations. Packet Reports are created for every selected packet of a Selection Process. Each Packet Report contains the already mentioned `selectionSequenceId` Information Element and a number of contiguous bytes of the packet [7]. Which packet sections are included is configurable and could be, for instance, realized by incorporating the `dataLinkFrameSection` Information Element into Packet Reports. Additionally, including several other packet properties is possible by configuring a template to include registered or enterprise-specific Information Elements in the Packet Reports. E.g., the inclusion of a timestamp taken during packet capture is helpful in many scenarios and will be discussed later.

Report Interpretations are used to provide access to additional statistics about the Metering Process itself. PSAMP describes several standard types of Report Interpretations [7]. These Report Interpretations provide information such as the accuracy of timestamps, the configuration of Selection Processes and Selectors, and packet counts at several points in the Metering Process. The information included in the Report Interpretations is also flexible and defined by templates. Altogether, these statistics can assist a network function in interpreting the Packet Reports' data and offer new insights. For example, the Selection Sequence Statistics Report Interpretation provides values of counters that count the number of packets selected by Selectors of a Selection Process. This information can be used to infer the packet rate of the corresponding Selected Packet Stream.

Consequently, PSAMP supports the incorporation of various Information Elements, including packet properties and content, in the selection results referred to as Packet Reports. Additionally, Packet Report Interpretations provide statistics about the configuration and components of the Metering Process. Within the PSAMP context, the combination of Packet Reports and Report Interpretations is known as the Report Stream.

R6 - Low-Latency Processing
The selection result delay issue persists in many employments of the IPFIX architecture. The delay originates from a series of timeout mechanisms, including one in the Exporting Process that is most relevant [17]. The Exporting Process is typically responsible for the

aggregation of Flow Records. These Flow Records expire after a defined timeout (15 s-60 s) and are exported afterward. This results in a significant delay between observing a Flow packet and exporting related selection results, which directly conflicts with the **Low-Latency Processing** requirement. PSAMP Packet Reports, on the other hand, are exported immediately upon packet observation. This feature presents another reason why the concept of Flow Records will be entirely replaced by immediate Packet Reports in this architecture.

R7 - Distributed Collection
IPFIX and also PSAMP support the distribution of packet observation. Packets are observed and filtered at the PSAMP Device located at the Observation Point. The Report Stream, containing the selection results of the observed packets, is exported by the PSAMP Device's Exporting Process. The data is exported to another PSAMP component, the Collector. A Collector can collect Report Streams of one or more PSAMP Devices. In that way, selection outcomes from various locations in the network are centralized within the Collector component from where a network function could utilize the data.

In conclusion, a packet selection architecture based on PSAMP can meet the first eight architecture requirements while remaining standard compliant. As a result, the architecture can be classified as a standardized solution, also fulfilling requirement **R9 - Standardized Solution**.

5.2 Placement of the Network Function within the Architecture

While the components and protocols to use are clarified in the previous section, one last decision has to be made concerning the architectural layout. RFC 5470 [8] describes two possible placements for an application, e.g., a network function that requires packet selection results, within the IPFIX/PSAMP architecture. The network function can either be an external component with a connection to the Collecting Process (Figure 5.2 (1)) or a module that coexists on the Collector with the Collecting Process (Figure 5.2 (2)). The advantage of position one is the independence from all other components. That means the connection is more flexible, and connections to multiple Collectors would be architecturally possible. The disadvantage, however, is that the packet selection result data undergoes an extra step to be accessible to the network function. I.e., the Collector has to ensure data availability to the external application, e.g., by exporting

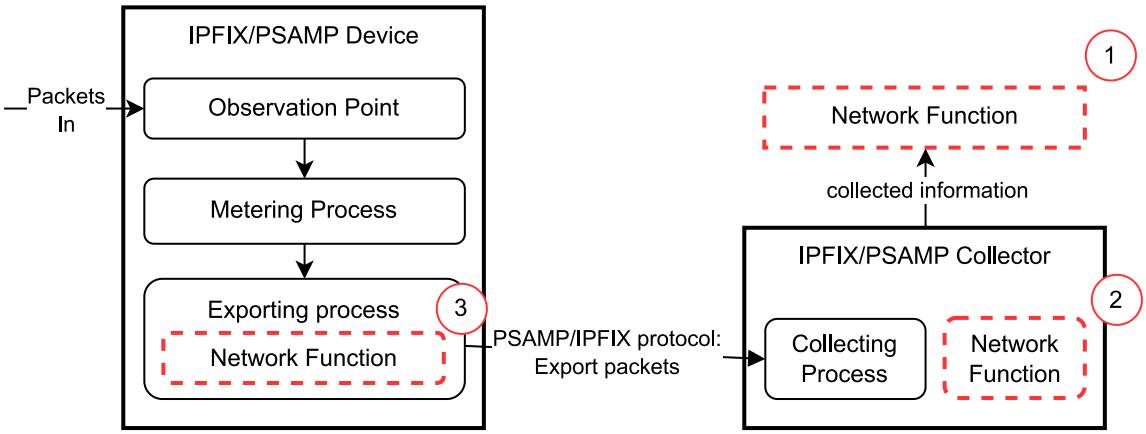

Figure 5.2: Placement options for the network function within the IPFIX/PSAMP architecture, position two achieves low result availability delay and facilitates distributed network functions

the data via TCP, which encompasses extra delay on data availability. Position two avoids this issue by neglecting the additional flexibility of position one.

Another placement option, not mentioned in the standards, was demonstrated in [17] by Hofstede et al. They tried to achieve real-time intrusion detection using an IPFIX Flow monitoring architecture. To overcome delays in the Collector and Exporting Process caused by Flow aggregation, they placed their traffic analysis application directly onto the IPFIX Device at the Exporting Process. While this approach significantly reduced delays, it is irrelevant to our proposed architecture, which does not involve Flow aggregation and, therefore, no such delays. Additionally, by placing the application inside the PSAMP Device, we would lose the possibility of **Distributed Collection (R7)**. Finally, we decided to use position two to ensure distributed collection while minimizing additional result availability delay and architecture complexity.

5.3 Architecture Description

The previous sections demonstrated all the decision-making involved in conceptualizing the new flexible packet selection architecture. Now, this section will outline the final structure of the architecture. The PSAMP components identified in the previous section are applied and connected to support the network function with packet selection functionality while achieving the architecture requirements.

The architecture consists of two main components: the PSAMP Device and the PSAMP Collector. The PSAMP Device contains an Observation Point, Metering Process, and Exporting Process; the PSAMP Collector includes a Collecting Process and the network function. The packet stream is observed at the Observation Point and handed to the

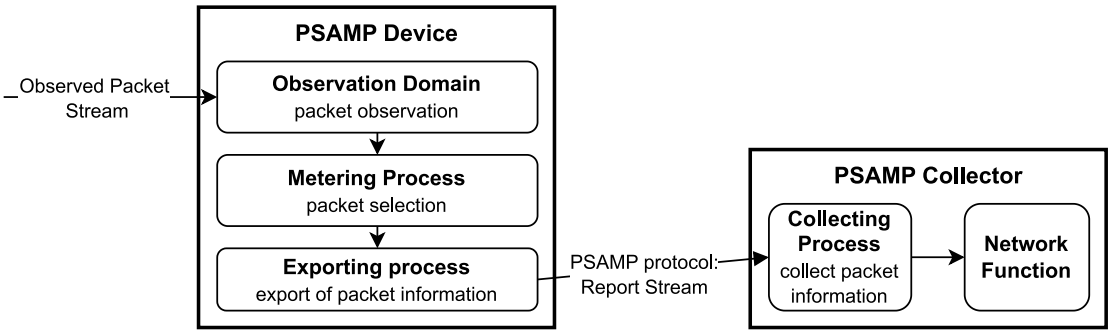

Figure 5.3: Final packet selection architecture composition

Metering Process for timestamping packet selection, and packet information extraction. The Metering Process results are exported as PSAMP Report Stream, using the Exporting Process, to the Collecting Process of the PSAMP Collector. The Collecting Process collects the export from one or potentially more Exporting Processes and forwards the packet selection results to the network function.

The overall architecture in its presented form complies with the IPFIX/PSAMP standards while meeting all the architecture requirements. The next focus lies on the network function use cases and how this packet selection architecture draft supports their functionalities.

Chapter 6

Design of the Network Functions

This chapter will explain how network functions effectively utilize the packet selection architecture with the example use cases of Network Health Monitoring and Tetrys.

6.1 Information Flow

To comprehend the functionality of the network functions within the architecture, it is crucial to examine the information flow within the PSAMP Collector, which also encompasses the network function. As Chapter 5 explains, the Metering Process exports Data Records (Packet Reports and Report Interpretations) encapsulated in PSAMP messages to the Collector.

To streamline the implementation of network functions, they are not designed to interpret PSAMP messages directly. Instead, the Collecting Process parses PSAMP messages and interprets them according to the PSAMP protocol [7]. Following interpretation, the Data Records must be converted to a format suitable for network function implementations.

The choice of a data format depends on the specific implementation architecture. However, the general goal is to select a single data format suitable for most types of network functions. E.g., in the Python implementation presented in the subsequent chapter, each Data Record is converted into a Python dictionary of key-value pairs. The keys represent Information Element names and the values their corresponding values. While not the most efficient format, this method is particularly advantageous because it is human-readable and easy to work with. Once converted, the Data Records remain divided into their two categories. However, these can be merged into a single data stream,

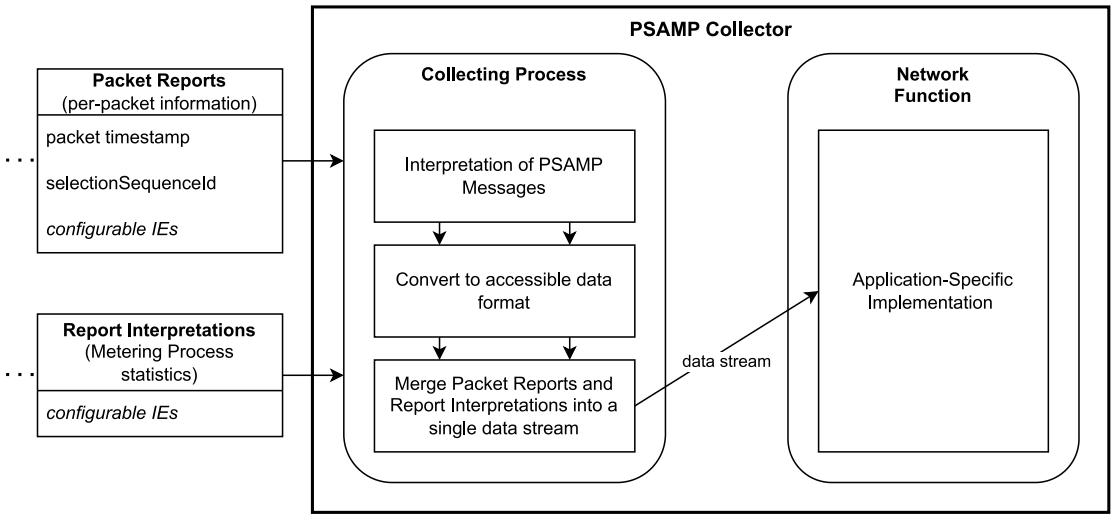

Figure 6.1: Information flow inside the PSAMP Collector of the packet selection architecture

as it is possible to infer from the Information Elements whether a Data Record is a Packet Report or a Report Interpretation. This distinction is possible because Packet Reports exclusively include several mandatory fields, such as the packet timestamp and the `selectionSequenceId`.

6.2 Network Health Monitoring

At its core, Network Health Monitoring involves measuring various Packet Flow characteristics to collect network statistics. These statistics are utilized to assess the network's health to enhance its stability, performance, and other essential attributes. This section will concentrate on the benefits of employing our packet selection architecture and outline how to achieve effective Network Health Monitoring using this architectural framework.

6.2.1 Advantages of the Packet Selection Architecture for Network Health Monitoring

The packet selection architecture enables the monitoring of Flow behavior by allowing the filtering of specific packets within network streams and collecting Flow-specific metrics. The definition of a Packet Flow can remain highly flexible and characterized by one or multiple Information Elements. Moreover, it is possible to observe numerous Packet Flows within a single network stream simultaneously by utilizing multiple Selection Processes.

Another advantage of the packet selection architecture is the facilitation of distributed network monitoring. Our architecture allows deploying multiple PSAMP Devices across the network, providing information to a single Network Health Monitor. This enables, e.g., the comparison of statistics for the same Packet Flow at different locations, offering insights into network behavior. For instance, Load Balancing techniques or bandwidth bottlenecks can be measured and evaluated.

The packet selection architecture also simplifies metric gathering. PSAMP permits extracting header fields and other properties from packets, which the network function can then utilize to calculate metrics. Consequently, the Network Health Monitor does not need to implement any packet parsing functionality, simplifying its implementation. Additionally, statistics about the Metering Process itself enable insights into Flow statistics.

Lastly, packet sampling allows network metrics to be approximated in high-bandwidth networks without the necessity for high-performing networking equipment. By analyzing only a defined fraction of the observed packets, statistics about Packet Flows can be approximated. This approach helps in managing the monitoring of large-scale networks effectively while keeping the performance requirements minimal.

6.2.2 Flow Metrics

This section will analyze various Flow metrics and demonstrate how they can be efficiently collected using the prototype architecture to its fullest potential. For each metric, we will explain how Selection Processes, Packet Reports, and Report Interpretations should be configured to facilitate the measurement of these metrics.

Packet Rate

There are two basic approaches to gathering the packet rates of a Flow. Suppose no other metrics requiring the export of Packet Reports are needed. In that case, it is sufficient only to use Report Interpretations to export the `selectorIdTotalPktsSelected` Information Element for each Selector, which describes the total count of packets that passed the Selector. For this purpose, PSAMP defines the Selection Sequence Statistics Report Interpretation, which includes all Selector counters of one Selection Process instance and the `selectorIdTotalPktsObserved`, representing the total count of packets observed at the Selection Process instance. Using the following formula, the Network Health Monitor can derive the packet rate from the counter differences between two Report Interpretations.

$$packetrate = \Delta packetcount / \Delta time \tag{6.1}$$

The more frequently Report Interpretations are exported, the more granular the packet rate measurements become.

The second method involves using Packet Reports. Counting the Packet Reports while considering the timestamps allows for a per-packet accurate packet rate metric to be derived. However, this approach sacrifices the simplicity of the first approach for more accurate results.

Packet Interarrival Times

Given a Packet Flow, the packet interarrival time can be measured by configuring a Selection Process to filter only for packets of this Packet Flow, for example, by filtering for source and destination-IP-address. To get the interarrival time between two packets, the timestamps of both packets are used to calculate their difference. The timestamp of selected packets in the PSAMP Device is included in every Packet Report, for example as `observationTimeMicroSeconds` Information Element. The Network Monitor can continuously calculate the difference between two consecutive packets, thus generating statistics about the packet interarrival time development over time.

One-Way Delay/ One-Way Packet Loss

Given a Packet Flow of packets with the same packet length sent at regular intervals, the one-way delay measurement [26] can be supported by the packet selection architecture. The clock of the packet sender and the PSAMP Device should be synchronized. The packet sender includes a timestamp in each packet. As such sender timestamps are not yet included as Information Elements, a new enterprise-specific Information Element must be configured before the measurement and supported by both the PSAMP Device and the Collector. The Selection Process should be configured to filter for the Packet Flow of interest. When a packet of the Flow arrives at the PSAMP Device, it receives a second timestamp. The sender timestamp is extracted and included, along with the second timestamp, in each Packet Report. Finally, the Network Health Monitor receives a stream of Packet Reports containing the two different timestamps, from which it can calculate the one-way delay for every packet. From this series of one-way delay measurements, one could also calculate the one-way delay variation metric [27].

If the packet sender includes a sequence number in the Flow packets, the Network Health Monitor could also derive the one-way packet loss [28]. The PSAMP Device now must be configured to extract and include the sequence number, e.g., as an enterprise-specific Information Element in its Packet Reports. By analyzing the stream of Packet Reports, the Network Health Monitor can now determine the one-way packet loss for the given packet stream. It only needs to keep track of the sequence numbers of packets that

arrived successfully. And for the sequence numbers that the monitor never received a packet for, it can derive, after a timeout, that the packet is lost.

Throughput
Given a stream of packets, the PSAMP Device can be configured to export the packet length of each selected packet. The Selection Process could be configured to filter for specific Packet Flows or the entire network stream. The Network Monitor receives the packet length, e.g., as `ethernetTotalLength` Information Element, along with each packet's packet timestamp in the form of Packet Reports. To calculate the total network throughput at any time interval, one can sum up all packet lengths of packets with corresponding timestamps within the interval.

The metrics mentioned in this section focus on those that can be measured using a single PSAMP Device due to their simplicity and relevance for Chapter 7. While the goal of this section is not to provide a comprehensive list of possible metrics, it aims to demonstrate the usability and versatility of the proposed packet selection architecture.

6.3 Network Coding: Tetrys

This section explores the potential of the PSAMP-based packet selection architecture for Tetrys, an on-the-fly Network Coding protocol. Tetrys is designed to transport delay-sensitive and loss-sensitive data over lossy networks. The transmission of coded packets allows Tetrys to recover from packet loss [11]. Depending on network conditions, Tetrys can dynamically adjust the redundancy introduced in the coded packets, defined by the redundancy ratio, to achieve the desired transmission reliability.

6.3.1 The Tetrys Architecture Concept

The objective is to ensure reliability for loss-sensitive Packet Flows within an Ethernet network stream between nodes A and B under dynamic network conditions and varying reliability requirements per Flow. Furthermore, dynamic inclusion or exclusion of Flows for Tetrys transmission is desired. We propose a control plane-managed solution that combines our packet selection architecture with a distributed Tetrys network function to address these challenges. Figure 6.2 depicts the proposed concept, which will be elaborated upon in the following.

To support this Tetrys-specific functionality, the Information Elements shown in Table 6.2 are used. The definition of two new enterprise-specific Information Elements listed in Table 6.1 is necessary to represent per-Flow packet loss and a per-Flow redundancy ratio setting for Tetrys.

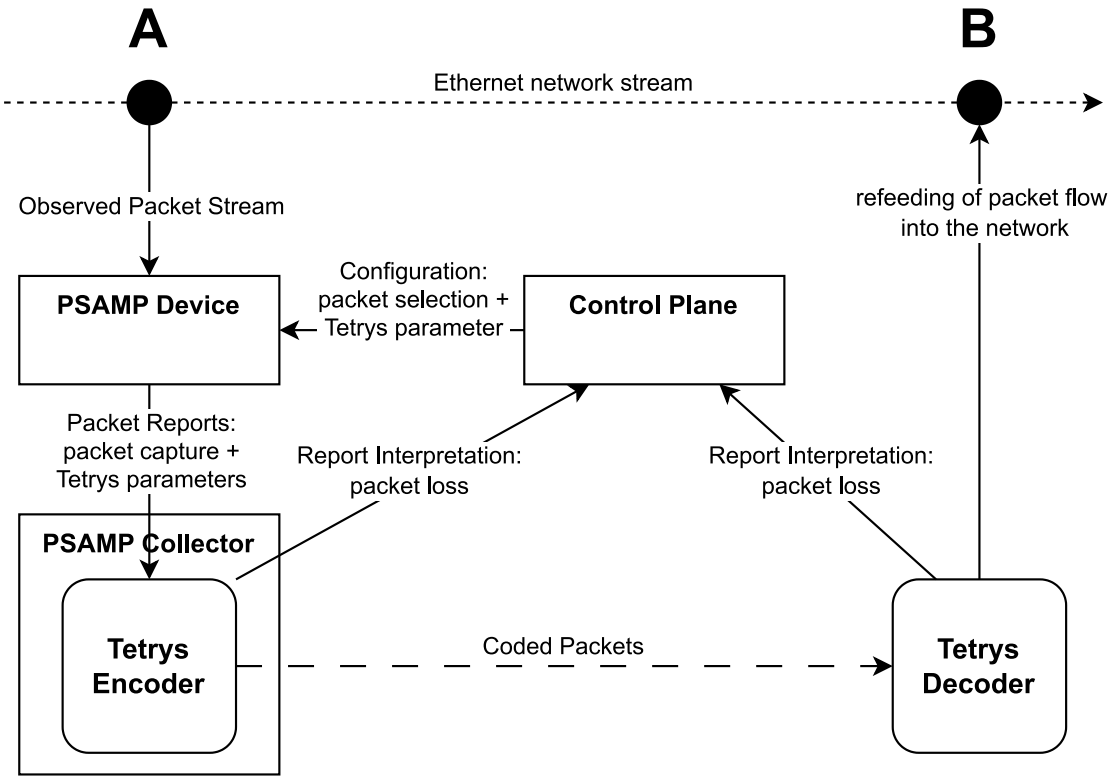

Figure 6.2: Tetrys Architecture Concept

Enterprise-Specific ID	Name	Description
1	packetLoss	Fraction of lost packets, expressed as percentage
2	redundancyRatio	Tetrys redundancy ratio for a Flow

Table 6.1: Enterprise-specific Information Elements for the Tetrys Architecture Concept

ID	Name	Description
148	flowId	An identifier of a Flow that is unique within an Observation Domain
315	dataLinkFrameSection	This Information Element carries n octets from the data link frame of a selected frame

Table 6.2: Existing Information Elements for the Tetrys Architecture Concept from the IANA IPFIX registry [9]

The control plane knows the loss-sensitive Packet Flows and configures the PSAMP Device at node A to classify them by their Flow Keys and to export corresponding Packet Reports to Tetrys. It is also possible to dynamically add or remove such Flows to adjust for changes in network traffic. Such a configuration can be remotely controlled using the Network Configuration Protocol (NETCONF), compatible with the PSAMP configuration models [25]. The control plane also configures Tetrys redundancy ratios for each loss-sensitive Packet Flow based on the network conditions and Flow reliability requirements. The Packet Reports exported from the PSAMP Device to Tetrys contain three Information Element values. The `dataLinkFrameSection` Information Element is included with a variable length value to represent the packet captures of selected Flow packets. Additionally, the `flowId` Information Element is used for Flow identification, and the new enterprise-specific `redundancyRatio` Information Element incorporates the desired redundancy configuration for the Flow.

The Tetrys network function comprises the Tetrys Encoder located at node A and the corresponding Tetrys Decoder at node B. The Tetrys Encoder incorporated in the PSAMP Collector receives Packet Reports from the PSAMP Device, containing all necessary information concerning Flow packets. First, the `redundancyRatio` value included in each of these Packet Reports is used to configure the per-Flow redundancy ratio of the encoder. Then, the Ethernet packet captures in the form of the `dataLinkFrameSection` Information Elements from the Packet Reports are coded and transmitted to the Tetrys Decoder. At the Tetrys Decoder, Flow packet captures are decoded from the coded packets and reinserted into the network at node B. During its operation, the Tetrys network function monitors the packet loss experienced during the transmission of coded packets. This packet loss is reported to the control plane by exporting regular Report Interpretations containing the `packetLoss` Information Element value.

The control plane, in turn, calculates the per-Flow Tetrys `redundancyRatios` from the reported packet loss and the per-Flow reliability requirements and configures the PSAMP Device accordingly to include the current redundancy ratios in the Packet Reports.

This architecture ensures reliable transmission of critical Packet Flows between nodes A and B under dynamic network conditions with varying degrees of packet loss and variable network traffic composition. The individual treatment of Flows guarantees the fulfillment of reliability requirements.

In the following chapter, we will delve into the implementation details of some of the introduced concepts, further showcasing the practical applications and effectiveness of the packet selection architecture in Network Health Monitoring and Network Coding.

Chapter 7

Implementation

This Chapter focuses on implementing the PSAMP Collector and the PSAMP Device, which are the two main components of the proposed architecture. The goal is to match the packet selection architecture as closely as possible, focusing on proof of concept rather than performance optimization. An implementation should be able to demonstrate and validate the functionality of our architecture successfully. First, we will assess the availability of suitable open-source software and determine whether a custom implementation will be required.

7.1 Analysis of available Implementations

7.1.1 PSAMP Collector Implementation

The previously discussed architectural layout of the PSAMP Collector is unique in its design. Various existing open-source IPFIX/PSAMP Collector software implementations are available that implement the Collecting Process component of the Collector [29]. These existing implementations could be used in conjunction with an externally implemented network function. However, as outlined in Chapter 5, an implementation that closely integrates the network function and Collecting Process, resulting in quick and seamless information transfer between them, is strongly favored.

Therefore, it is preferred to develop a custom Collector implementation, enabling a more accurate demonstration of the architecture's potential while maintaining consistency with the proposed architecture model. The Collector prototype is developed using Python, which is well-suited for rapid prototyping. Utilizing Python for the prototype

allows complete control over the system's information flow, including separate but connected components for the Collecting Process and network function.

7.1.2 PSAMP Device Implementation

The PSAMP Device, as outlined in the proposed architecture, is fully compliant with the IPFIX/PSAMP standards. Given that the IPFIX/PSAMP standards have been in place for several years, various open-source software solutions, commonly referred to as Flow Exporters, are available. An analysis of available software was conducted by Hofstede et al [29]. Beyond those mentioned in this paper, no other suitable open-source implementations are available.

PSAMP Device Implementation Requirements

A new set of requirements is needed to evaluate the available solutions further. The basic requirements for a software solution capable of performing all of the PSAMP Device functionality are outlined below:

R1: Packet Filtering - The software should support packet filtering capabilities, allowing at least filtering based on the IP 5-tuple. The packet selection logic configuration should adhere to the PSAMP packet selection concept, involving Selection Processes, Selectors, and so on.

R2: IPFIX Protocol Support - The software must be capable of exporting IPFIX/PSAMP protocol messages.

R3: PSAMP Packet Reports - Most Flow Exporters focus on exporting Flow Records, the most common use case for IPFIX/PSAMP. However, exporting per-packet PSAMP Packet Reports is essential to implement the proposed architecture.

R4: Packet Classification - The packet selection process should allow configuring multiple Selection Processes, to enable packet classification. This classification mechanism would, e.g., enable filtering for multiple Packet Flows within one network stream.

R5: PSAMP Report Interpretations - The capability to export PSAMP Report Interpretations alongside Packet Reports is desirable, as it provides an alternative information source about the internal selection behavior for network functions to use in addition to Packet Reports.

Hereby the first three requirements (**R1**-**R3**) are necessary, and the last two (**R4**-**R5**) are desirable to validate the extended functionality of the architecture. Packet sampling

and Hash-based Filtering are out of scope for the demonstration and are, therefore, no requirements.

The first step in the implementation process of the PSAMP Device is evaluating available solutions to determine the degree to which custom software development is necessary. The aim is to find a solution that closely matches the defined architecture while prioritizing proof of concept over performance. The most suitable software can be identified by evaluating existing solutions against the requirements outlined above.

Evaluation of existing Open Source Software

In this section, we will analyze the potential of each presented solution for implementing the PSAMP Device functionality based on the defined requirements (**R1**-**R5**). The solutions examined include nprobe, pmacct, Vermont, and YAF. An overview of the requirements analysis results for each candidate is depicted in Table 7.1.

	nprobe	**pmacct**	**QoF**	**Vermont**	**YAF**
R1: Packet Filtering	(✓)	(✓)	(✓)	(✓)	×
R2: IPFIX Protocol Support	✓	✓	✓	✓	✓
R3: PSAMP Packet Reports	×	×	×	✓	✓
R4: Packet Classification	×	×	×	×	×
R5: PSAMP Report Interpretations	×	×	(✓)	×	×

Table 7.1: Comparison of PSAMP Device implementations (PSAMP Device requirement analysis), only Vermont fulfills all crucial requirements **R1-R3**

nprobe does not support Packet Reports, as it focuses solely on Flow aggregation [30]. Consequently, nprobe does not fulfill the essential requirements for our use case even though it offers packet filtering capabilities utilizing Berkeley Packet Filters (BPF).

pmacct enables Flow aggregation [31] while utilizing packet filters in regular expressions and hooks for external applications. Pmacct does not provide PSAMP export functionality, despite supporting the IPFIX protocol. Given these limitations, pmacct does not meet the critical requirements for our proposed architecture.

QoF offers packet filtering capabilities based on the Berkeley Packet Filter (BPF), but these do not extend to packet classification. While QoF supports IPFIX Flow Record export, it does not provide PSAMP Packet Reports, which is one of the critical requirements for our use case. QoF offers exporting information about its internal processes as IPFIX Options Template Records, the IPFIX equivalent to Report Interpretations [32].

Vermont provides IPFIX protocol support and packet filtering options [2]. It also supports Packet Reports with the inclusion of a PSAMP export module. Vermont can

fulfill the critical requirements, but it lacks support for Packet Report Interpretations and has limited filter capabilities. Due to these limitations, packet classification is not feasible in Vermont.

YAF can convert each observed packet into a single Packet Flow, equivalent to Packet Reports. However, YAF does not offer packet selection or filtering features [33]. As a result, YAF is not a suitable choice for implementing the PSAMP Device functionality in the proposed architecture.

Based on the analysis, none of the examined solutions fully satisfy the requirements for implementing the PSAMP Device functionality in our proposed architecture. Vermont appears to be the closest match, but its limitations regarding Packet Report Interpretations and packet classification make it less than ideal. The following section analyzes the shortcomings of Vermont to determine if it is necessary to develop a custom PSAMP Device.

Vermont as a PSAMP Device Implementation

Vermont is an open-source project that aims to provide IPFIX and PSAMP functionality [2]. In this section, we will discuss Vermont's features and limitations in terms of implementing the PSAMP Device functionality and analyze the feasibility of extending Vermont to address these limitations.

Vermont Features: Vermont provides several features that align with the desired PSAMP Device functionality. It can process data from PCAP (Packet Capture) files or network interfaces through its Observer component. Vermont offers various packet selection techniques through its Packet Filter component, although they differ from the PSAMP Selection Process concept. These filters include regular expressions, IP address filters, string matches, and sampling techniques. Furthermore, Vermont supports the export of Information Elements within Packet Reports through its PSAMP Exporter component.

Limitations of Vermont: Despite its advantageous features, Vermont has several limitations that prevent it from effectively implementing the PSAMP Device functionality, as outlined in the defined requirements. In certain aspects, Vermont does not fully conform to the PSAMP standards. For instance, it does not incorporate the Information Element `SelectionSequenceId` into Packet Reports, and it only allows the definition of a single Selection Process, i.e., the Packet Filter component. This results in an inability to classify network streams as described in Chapter 5. Additionally, configuring packet filters in Vermont is not based on PSAMP and is not as straightforward as initially de-

scribed in the architecture definition. Although it might be feasible to create filters for packet properties equivalent to PSAMP filters using regular expressions, the complexity involved in crafting such expressions makes this approach impractical. Moreover, the IP address filter operates as a whitelist, meaning that any packets with matching source or destination IP addresses will be selected. Using the IP address filter, it is not possible to create a composite filter that selects packets based on pairs of source and destination IP addresses.

Considerations regarding a Vermont Extension: One may consider extending Vermont to overcome its limitations and better align with the PSAMP Device functionality requirements. However, there are several reasons why this might not be the most suitable approach for this thesis. Extending Vermont would be time-intensive, given its complexity and the need to address multiple limitations. The lack of prior familiarity with the project would increase the time and effort required for extension. Additionally, it would be necessary to develop a new packet selection component or extend the existing filter to add filtering for packet properties. Although adding additional Information Elements might not be a significant challenge, incorporating the `SelectionSequenceId` would require modifications across multiple program parts, including the Packet Filter, PSAMP exporter, and internal information management.

Given these reasons, extending Vermont is not ideal as a proof of concept for this thesis but could be considered for future work based on the findings in this thesis. In the next chapter, we will evaluate limited use cases that can be performed using Vermont, despite its limitations, combined with the custom PSAMP Collector implementation.

Conclusion

Given the analysis of the available solutions (nprobe, pmacct, QoF, Vermont, and YAF), it becomes evident that none of them can fully satisfy the requirements for implementing the PSAMP Device functionality in our proposed architecture. Therefore, to demonstrate and validate all the desired packet selection architecture functionality, the development of a custom PSAMP Device implementation is necessary. The prototype, analogous to the PSAMP Collector, will be implemented in Python. This custom implementation will ensure that all requirements are met while providing a proof of concept that validates the functionality of the proposed architecture.

7.2 Python Prototype Implementations of PSAMP Device and Collector

The previous section demonstrated that it is necessary to develop custom software prototypes to validate the packet selection architecture's functionality. This section explains the relevant details of both implementations and describes how the various concepts of the packet selection architecture have been translated into code.

7.2.1 Overview

The architecture of the Python implementation of our system directly follows the packet selection architecture's composition and is therefore primarily composed of two components: `psamp_device.py` and `psamp_collector.py`.

The former, `psamp_device.py`, encompasses all functionalities of the PSAMP Device, including packet observation at the network interface, timestamping, packet filtering based on pre-configured Selection Processes, and exporting PSAMP protocol messages to the `psamp_collector.py`. The `psamp_device.py` component relies on the `psamp_config_state.py` module, which houses the classes responsible for representing the configuration instance parsed from the `config.json` file. Instantiating these classes generates a data structure that maintains the internal state and configuration for the PSAMP Device functions called `config_state`. We will first discuss the configuration and PSAMP Device state structure, as it serves as the foundation for the `psamp_device.py` component.

The second component, `psamp_collector.py`, establishes a network connection with a `psamp_device.py` component, receiving PSAMP protocol messages containing the outcomes of packet selections. This component is capable of parsing and interpreting these messages, and it executes specified functions on each PSAMP Data Record it receives. The particular function executed depends on the implemented network function type.

7.2.2 Development Process and Implementation

This section describes the Python implementation of the packet selection architecture in detail. The description follows the order in which the various parts were developed.

Configuration

The configuration component is the core of the PSAMP Device implementation. It is responsible for representing both the configuration and the internal state maintained by the PSAMP Device. The configuration is based on the configuration data module for IPFIX and PSAMP Devices presented in RFC 6728 [25]. The configuration file serves

as the configuration input for a Python class structure, which manages runtime access to both the configuration and the internal state. As file format for the configuration, JSON was chosen in favor of XML because of a simpler and more compact syntax which simplifies reading, understanding, and manual editing.

The configuration file - `config.json`: The PSAMP Device is configurable by editing the `config.json` file. The file structure was derived from the `ietf-ipfix-psamp@2012-09-05` YANG module in RFC 6728 [25]. To get a good overview of the YANG module, including the data types of individual fields, the tree representation available in the Python-based YANG tool Pyang [34] offers a compact human-readable format (Listing 7.1). The derivation of the `config.json` file is demonstrated in the following:

```
+--rw observationPoint* [name] {meter}?
|  +--rw name                    nameType
|  +--ro observationPointId?     uint32
|  +--rw observationDomainId     uint32
|  +--rw ifName*                 ifNameType
|  +--rw ifIndex*                uint32
|  +--rw entPhysicalName*        string
|  +--rw entPhysicalIndex*       uint32
|  +--rw direction?              direction
|  +--rw selectionProcess*       -> /ipfix/selectionProcess/name
```

Listing 7.1: PSAMP YANG configuration module tree representation

1. Pyang was used to generate a sample XML skeleton for the instance of the YANG module with `pyang -f sample-xml-skeleton -sample-xml-skeleton-defaults`. Listing 7.2 shows a snippet of the Observation Point part of the XML file resulting from this step.

2. This XML file was then translated to JSON using the built-in Python JSON module [35] and the Python `xmltodict` module [36] as shown in Listing 7.3. Here `xml_yang-config` is a String containing the XML skeleton result of the last step.

3. The resulting `json_yang-config` contains the JSON YANG module structure. The JSON file in its final form, filled with the mandatory data entries, is depicted in Listing 7.4.

4. The configuration is mostly left unchanged; only unnecessary configuration options are omitted. However, besides the configuration parameters described in the YANG module, the implementation requires several additional configuration

```
<observationPoint>
  <name/>
  <observationPointId/>
  <observationDomainId/>
  <ifName>
    <!-- # entries: 0.. -->
  </ifName>
  <ifIndex>
    <!-- # entries: 0.. -->
  </ifIndex>
  <entPhysicalName>
    <!-- # entries: 0.. -->
  </entPhysicalName>
  <entPhysicalIndex>
    <!-- # entries: 0.. -->
  </entPhysicalIndex>
  <direction>both</direction>
  <selectionProcess>
    <!-- # entries: 0.. -->
  </selectionProcess>
</observationPoint>
```

Listing 7.2: PSAMP YANG configuration module XML skeleton

```
import xmltodict
import json
# Convert XML to dictionary
xml_dict = xmltodict.parse(xml_yang-config)
# Convert dictionary to JSON
json_yang-config = json.dumps(xml_dict)
```

Listing 7.3: Conversion of XML skeleton to JSON using Python

```
"observationPoint": [
            {
                "name": "mainObservationPoint",
                "observationPointId": "1",
                "observationDomainId": "1",
                "direction": "Ingress",
                "selectionProcess": "SelectP1",
                "ifIndex": "1"
            }
]
```

Listing 7.4: PSAMP YANG configuration module as JSON skeleton

parameters. These additional parameters are also included in the JSON file and marked with the "custom" prefix and will be elaborated in subsequent sections.

The Python JSON module [35] is used to parse the `config.json` file. The resulting object is then passed to the class structure in the `psamp_config_state.py` module, leading to the instantiation of a `config_state` object.

Additional internal ID handling mechanisms are integrated into the module for the Selection Processes and Selectors. These mechanisms ensure that both Selection Processes and Selectors are each assigned incremental IDs upon initialization of the `config_state` object. Each Selection Process contains exactly one Selection Sequence instance in accordance with the simplified packet selection architecture. As a result, every Selection Process is assigned a single `selectionSequenceId` Information Element. As specified in the Information Element description, this ID identifies the Observation Point and the sequence of Selectors. This is fulfilled by including the `observationPointId` Information Element, truncated to 16 bits, and an implementation-internal `selectionProcessId`, a 48-bit unsigned integer identifying a Selection Process. The 64-bit `selectionSequenceId` is generated by concatenating `observatonPointId` and `selectionProcessId`. Implementing the `selectionSequenceId` is crucial for including relations from Report Interpretations and Packet Reports to Selection Processes in a PSAMP-compliant manner. This relation enables packet classification.

PSAMP Device Implementation

The PSAMP Device is structured into two concurrently running threads: Packet Parser and Exporting Process. The role of the first thread is to parse the Observed Packet Stream, apply packet selection to it, and then push the positive selection results as a producer to a synchronized queue. The Exporting Process generates Packet Reports and other PSAMP protocol messages from the items it consumes from the same synchronized queue. The PSAMP protocol messages are exported to the `psamp_collector.py` via TCP. On initialization, the PSAMP Device parses the `config.json` file and, based on it, instantiates a `config_state` object from the already discussed `psamp_config_state.py` module. After that, it starts both Python threads.

Packet Parser - Timestamping and Packet Selection: The packet parser thread constantly listens to the Observed Packet Stream on a raw network interface. All arriving packets are parsed individually, including the readout of their hardware timestamps (NTP Timestamp format [37]). To prepare the packets for packet selection, the raw Ethernet packets are parsed into a `scapy.layers.l2.Ether` object [38]. The Python program Scapy [39] provides very convenient packet decoding functionality utilized

throughout the packet selection stage. It is, for example, used to extract packet header values required for packet filtering. The packet object, together with the `config_state` object, is handed to the `packet_selection()` function, which performs the configured selection operations and returns the selection result. The `packet_selection()` function generates selection results for each configured Selection Process individually. The results are represented using a list of dictionaries where every dictionary stands for the selection result of a single Selection Process. The dictionary entries follow this scheme:

```
{IPFIX Information Element name: value}
```

If such a dictionary, referred to as `metadata_dict`, is empty, the selection result of a Selection Process is negative, i.e., the packet is not selected. For every positive selection result, the `metadata_dict` includes an entry for `selectionSequenceId` referring to the involved Selection Process. Furthermore, metadata (e.g. hash-values) may be generated and added to the `metadata_dict` by a Selector of the Selection Process. Based on the configuration, the function sequentially applies the defined Selectors for every Selection Process. If the result of one of the Selectors is negative, the Selection Process stops and returns a negative result. If a Selector is applied, its counters in the `config_state` object are incremented accordingly (i.e. `selectorIdTotalPktsObserved`, `selectorIdTotalPktsSelected` [9]). The implementation supports two Selectors: the self-explaining `select_all` Selector and the packet filter `property_matching`.

The packet selection is performed for each packet using the `packet_selection()` function. Only if at least one selection result for a packet is positive, the packet, its hardware timestamp, and the `metadata_dict` list (i.e. selection results) are pushed onto the synchronized queue. This FIFO queue object is shared between both threads. It is an instance of the `Queue` class offered by the Python queue module, part of the Python Standard Library [35].

Exporting Process - Data Export: The Exporting Process thread maintains a TCP connection to the PSAMP Collector component, constantly exporting PSAMP protocol messages. These messages can be either Packet Reports based on the data consumed from the shared queue object or Packet Report Interpretations and Template Records generated based on the `config_state` object.

First, the Exporting Process connects as a TCP client to the PSAMP Collector's TCP server. Once the connection is established, as the first PSAMP protocol message, it transmits the Template Records necessary for the receiver to correctly interpret the Packet Reports and Report Interpretations. As the Template Records, due to the TCP connection, are guaranteed to arrive at the TCP server, they will not be retransmitted

[40]. Template Records are generated from their definition within the `tcpExporter` component of the `config_state` object.

Report Interpretations contain additional statistics about the PSAMP Device, such as counters. These statistics are maintained in the `config_state` object. The interval at which the Report Interpretations are exported is defined in the `tcpExporter` component within the `config_state` object as custom configuration option `customPacketReportInterpretationInterval`. The information that is included in Packet Report Interpretations depends on the configured templates.

Packet Reports are generated and exported when an item is available for consumption at the synchronized queue. As already discussed, a queue item contains the packet object, timestamp, and a list of `metadata_dicts` as a representation of the packet selection results. Each positive selection outcome results in an individual Packet Report, which typically contains the packet timestamp as `observationTimeNanoSeconds` and the `selectionSequenceId` [7]. To each Packet Report, an Information Element representing a number of contiguous bytes from the packet should be added [7]. This is required initially so a Packet Report can be related to a packet; however, this relation is not required in our prototype. Therefore, such Information Elements are only added if necessary for the network function. The required Information Element values are extracted from the `config_state` and packet object.

PSAMP Collector Implementation

The second main component of the prototype implementation is the PSAMP Collector, which also consists of two concurrent threads: Collecting Process and Network Function. It also maintains its own instance of the `config_state` object to manage templates and configure the connection to the PSAMP Device.

Collecting Process: The Collecting Process acts as a TCP server, receiving PSAMP protocol messages from the PSAMP Device via TCP. The decision to have the Collecting Process as a server was made to enable the connection of multiple PSAMP Device TCP clients if desired. The PSAMP protocol messages received as a byte stream on the TCP socket are dissected into individual messages using the message length field and then pushed to a synchronized FIFO queue based on the Python `queue` module [35].

Network Function: The network function is implemented inside the Network Function thread, which processes the PSAMP messages from the synchronized queue one at a time. For that purpose, the PSAMP Collector implementation offers functionality to parse PSAMP messages into an easily accessible data representation, the `metadata_dicts`

introduced in the PSAMP Device implementation. Each PSAMP protocol message on the synchronized queue comprises one or more sets, each containing one or more Data Records or Template Records. When a Template Record is parsed, it is added to the templates stored in the `config_state` object. These saved templates are subsequently used for interpreting Data Records. Each Data Record, either Packet Report or Report Interpretation, results in a `metadata_dict` that is ultimately forwarded to the network function implementation.

With this setup, the Network Function component can access all the data extracted and filtered from the Observed Packet Stream at the PSAMP Device. These Data Records can then be used to perform the necessary network function depending on the use case. The demonstrated use cases include a Network Health Monitor that supports calculating Packet Flow metrics as described in Chapter 6.

7.3 Conclusion

In the first section, we determined from a set of defined requirements that no currently available open-source implementation satisfies the requirements of our packet selection architecture. Vermont is the only candidate that supports the crucial export of Packet Reports and is thus selected for testing. However, the primary takeaway from this analysis was that a Python prototype for both the PSAMP Device and the PSAMP Collector is necessary to demonstrate the full potential of the packet selection architecture. The discussed development process firmly adheres to the PSAMP and IPFIX RFC standards. The Python prototype's internal architecture of the PSAMP Device and the PSAMP Collector matches the packet selection architecture. A practical data structure, the `metadata_dict`, is introduced to represent packet selection results internally. Lastly, a network function implemented at the PSAMP Collector receives packet result data as `metadata_dicts`. As a result, we now have two functional prototype architectures: one using the Python PSAMP Device and Python PSAMP Collector, and the other consisting of Vermont in conjunction with the Python PSAMP Collector.

Chapter 8

Evaluation

This chapter explores test scenarios for the packet selection architecture. The network function use cases elaborated on previously are applied to support a computer vision application. Initially, the Network Health Monitoring use case is implemented and tested using the Python and Vermont prototypes. Subsequently, an application of the Tetrys Architecture Concept defined in Chapter 6 is demonstrated. Lastly, the flexibility of the packet selection architecture towards network functions is evaluated.

8.1 Test Methodology

Figure 8.1 and 8.2 illustrate the experiment setups for the Python and Vermont prototypes on the test bed environment provided by the Chair of Network Architectures and Services at the Technical University of Munich. A pre-generated PCAP file is replayed on a physical network link connecting Test Node 1 and Test Node 2 using Tcpreplay [41] to ensure experiment reproducibility. The PSAMP Device (Vermont/ Python `psamp_device.py`), running on Test Node 2, listens to this hardware network interface and performs packet selection according to the configuration file. The resulting PSAMP Report Stream is then exported to the PSAMP Collector Python prototype. Within the Network Function component of the PSAMP Collector, a Network Health Monitor is deployed to measure various Flow metrics.

The PCAP file, representing a test scenario, is generated using the offline traffic generator developed by Kilian Holzinger [42]. This generator allows the generation of network traffic consisting of various Packet Flows resulting from simulated cameras. The length of a test scenario is 61 seconds, of which the first second is neglected in the measurements to avoid the startup behavior of the implementations impacting the measurements.

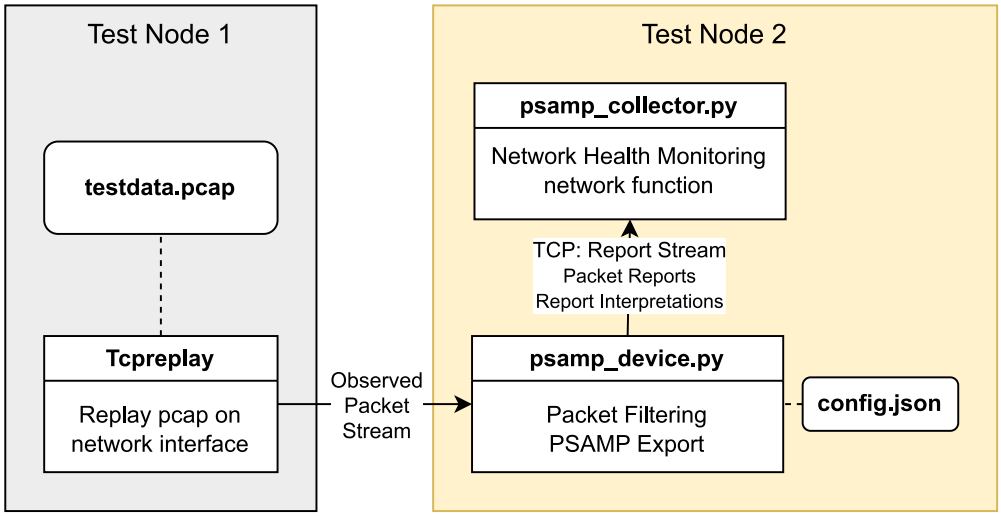

Figure 8.1: Python prototype: Test setup on the test bed environment

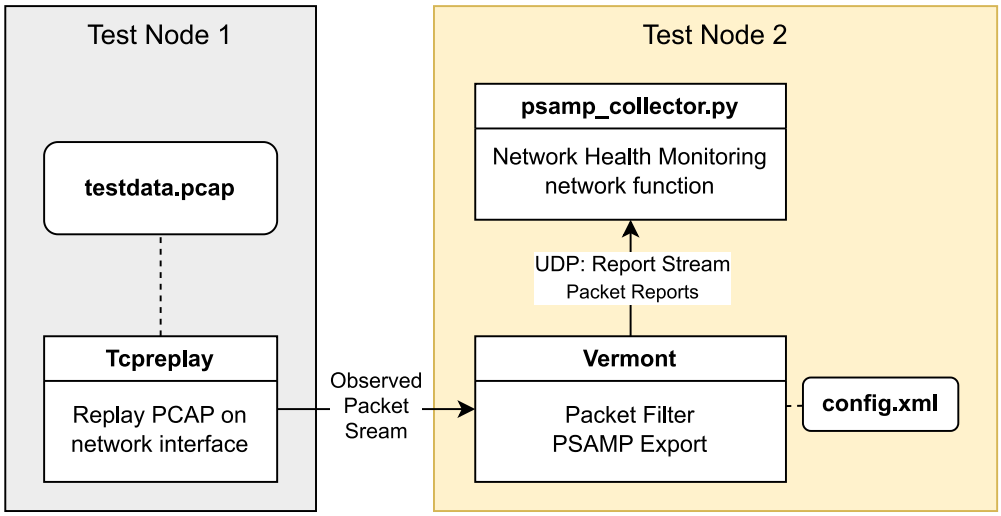

Figure 8.2: Vermont prototype: Test setup on the test bed environment

	Camera 1	Camera 2
Transport Protocol	UDP	UDP
Image Size (B)	1916	1458
Ethernet Frames		
Frame 1 Size (B)	1500	1500
Frame 1 IAT (ms)	33.3	33.3
Frame 2 Size (B)	500	-
Frame 2 IAT (ms)	0	-
Image Rate (fps)	30	30
Packet Rate (pps)	60	30
Throughput (Mbit/s)	0.48	0.36

Table 8.1: Test scenario: Dual Camera, simulation of network traffic from a computer vision application

8.1.1 Test Scenario - Dual Camera

To evaluate both network function use cases, Network Health Monitoring and Tetrys, we must define a base scenario where both use cases can demonstrate their functionality.

This scenario involves implementing a critical computer vision application that requires regular camera data transmission over the network. In our scenario, two cameras periodically send image data across the network. The Network Health Monitor is employed to monitor the packet rate, packet interarrival times, and throughput of both Camera Flows. In the Tetrys use case, the transmission reliability of the Camera Flows is maintained despite a lossy network.

To represent this test scenario, the traffic generator is configured according to Table 8.1 to include two Packet Flows representing periodic image exports from the two cameras into the PCAP file. Camera 1 exports images that are fragmented into two Ethernet Frames. Camera 2 exports smaller images that fit into a single Ethernet frame. Consequently, Camera 1 sends a burst of two packets 30 times per second, and Camera 2 a single packet 30 times per second. Both Flows are identified by their source IP address or their source port.

8.1.2 Reproducibility of Test Results

As previously mentioned, all tests are designed to be reproducible using the test bed environment at the Chair of Network Services and Architectures at the Technical University of Munich. This controlled test environment minimizes any variations in hardware configurations that could otherwise impact the results. Additionally, each test run is executed on a fresh installation of Debian Bullseye. This guarantees a consistent start-

ing point for each test, eliminating any potential interference from previous test runs or configuration changes.

To facilitate the automated execution of tests, a POS (Plain Orchestrating Service) [43] script is employed. This script executes synchronized test scripts on both test nodes simultaneously. The exact procedure for reproducing the tests is documented in the Readme file in the thesis GitLab repository [44].

8.2 Network Health Monitoring Tests

In this section, we will explore the behavior and performance of a Network Health Monitoring network function utilizing the packet selection architecture. Actual test cases will be demonstrated with the implementations introduced in Chapter 7. The objective of the diverse test cases is to validate the flexible packet selection capabilities of the packet selection architecture and its real-world application. Additionally, the capabilities and performance of the Vermont prototype are evaluated against the Python prototype. Throughout the tests, it is expected that the Network Health Monitor can correctly measure the Packet Flow properties as defined in Table 8.1

8.2.1 Packet Rate

The packet rate test cases will compare two approaches to calculate the packet rate of Packet Flows using the Python prototype test setup. The theory behind deriving packet rates from Packet Reports and Report Interpretations is covered in Chapter 6. The test cases utilize the Dual Camera test scenario. The two Camera Flows are filtered and classified at the PSAMP Device according to their IP 5-tuple. Each of the two configured Selection Processes in the PSAMP Device filters for packets of one of both Camera Flows.

Test Case 1 - Report Interpretation-based

The first test will determine the packet rate from the packet counters included in the Selection Sequence Statistics Report Interpretation [7].

Configuration: The configured template of this Report Interpretation is shown in Listing 8.1. In this case, the `selectionSequenceId` Information Element serves as a scope field, determining the scope to which the other Information Elements of this Report Interpretation belong, specifically, a particular Selection Process. The PSAMP Device exports this Report Interpretation for every Selection Process at a defined interval. This export interval defines the granularity of the resulting packet rate measurements.

```
"selectionSequenceStatisticsReportInterpretation": {
    "observationDomainId": "0", "templateId": "257", "setId": "3",
    "field": [
        {"ieName": "selectionSequenceId",
        "ieLength": "4", "isScope": true},
        {"ieName": "selectorIdTotalPktsObserved", "ieLength": "4"},
        {"ieName": "selectorIdTotalPktsSelected", "ieLength": "4"},
        {"ieName": "selectorIdTotalPktsSelected", "ieLength": "4"},
        {"ieName": "selectorIdTotalPktsSelected", "ieLength": "4"},
        {"ieName": "selectorIdTotalPktsSelected", "ieLength": "4"},
        {"ieName": "selectorIdTotalPktsSelected", "ieLength": "4"}]
}
```

Listing 8.1: Selection Sequence Statistics Report Interpretation template configuration to export counters of Selectors

Including a timestamp in this Report Interpretation would enable determining the most accurate packet rate measurements from both timestamp and packet counters. However, including timestamps in Report Interpretations is not PSAMP compliant and is therefore omitted in favor of staying compliant.

As an alternative, timestamps are taken upon data arrival at the Network Health Monitor, which is expected to be pretty inconsistent depending on the network environment and the load on the Network Health Monitor. By monitoring the changes in packet counters and correlating them with the corresponding timestamps, we can approximate the packet rates of the Camera Flows.

Test Result 1: Figure 8.3 displays the test run results using the described configuration of the test bed and software at an export interval of 0.2 s. The PSAMP Device does not export any Packet Reports in this case. Therefore, the Network Health Monitor is under a very low load of just 5 pps. The two graphs present the calculated packet rates at the Network Health Monitor. The average packet rates are pretty accurate (Table 8.1), with 59.94 pps for Camera 1 and 29.97 pps for Camera 2. Nevertheless, both graphs exhibit significant fluctuations of up to 6 pps from the average, with a mean error deviation of 2.06 pps (Camera 1) and 1.96 pps (Camera 2). These fluctuations are attributed to the previously discussed timestamping issue, where timestamps are taken at the Network Health Monitor instead of the PSAMP Device. The packet counters, originating from the PSAMP Device, are sent as PSAMP messages over the network to the network function, where they are parsed and timestamped. The transmission and parsing cause varying transmission delays that lead to inaccurate timestamps, affecting the packet rate metric. This fluctuation remains consistent across multiple test runs.

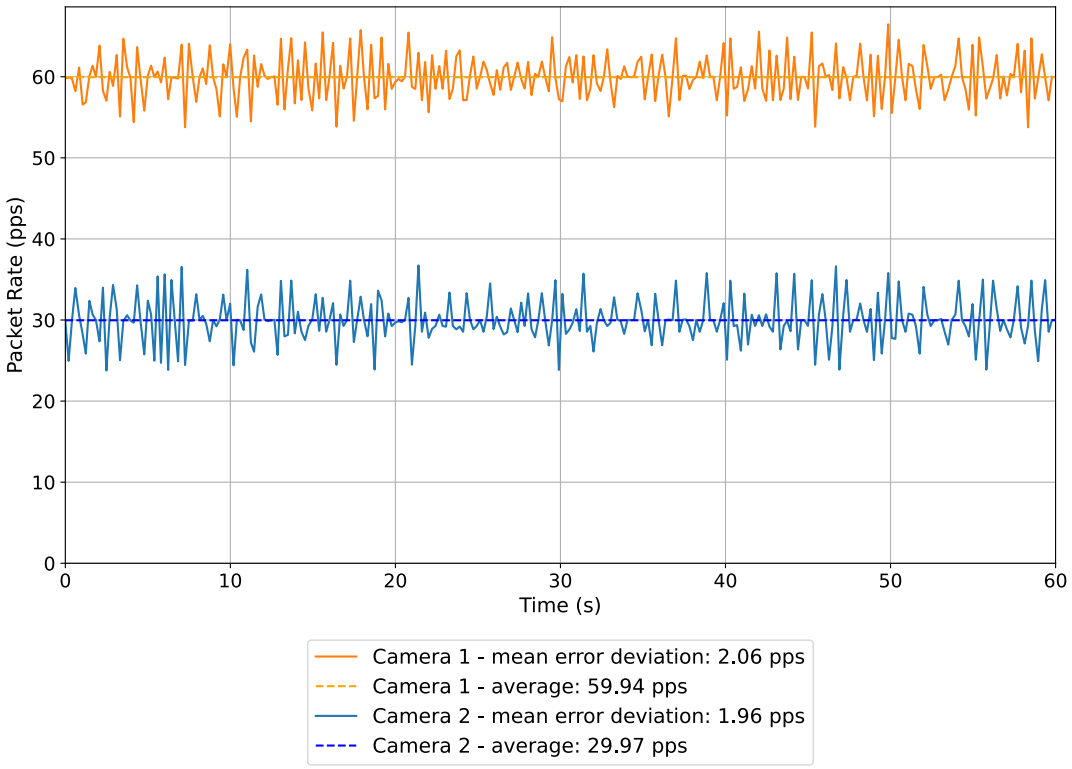

Figure 8.3: Python prototype: packet rate measurement (Report Interpretation-based, export interval: 0.2 s, no Packet Report export)

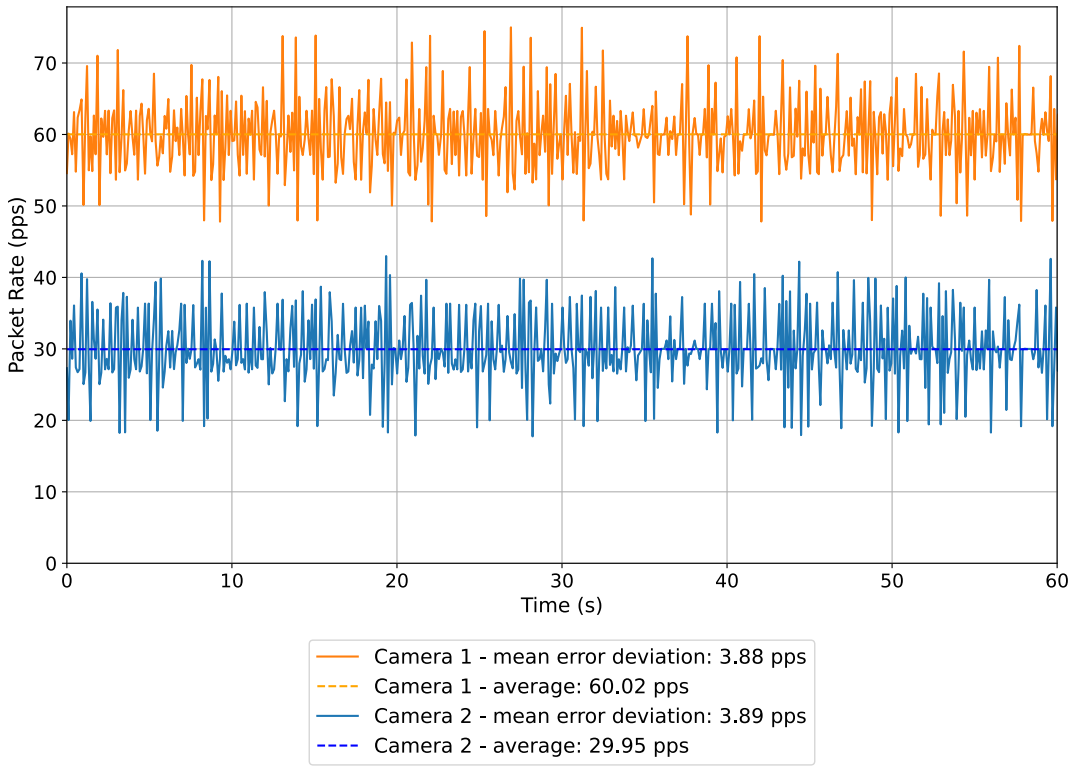

Figure 8.4: Python prototype: packet rate measurement (Report Interpretation-based, export interval: 0.1 s, no Packet Report export), the decrease of the export interval from 0.2 s to 0.1 s reduces the accuracy

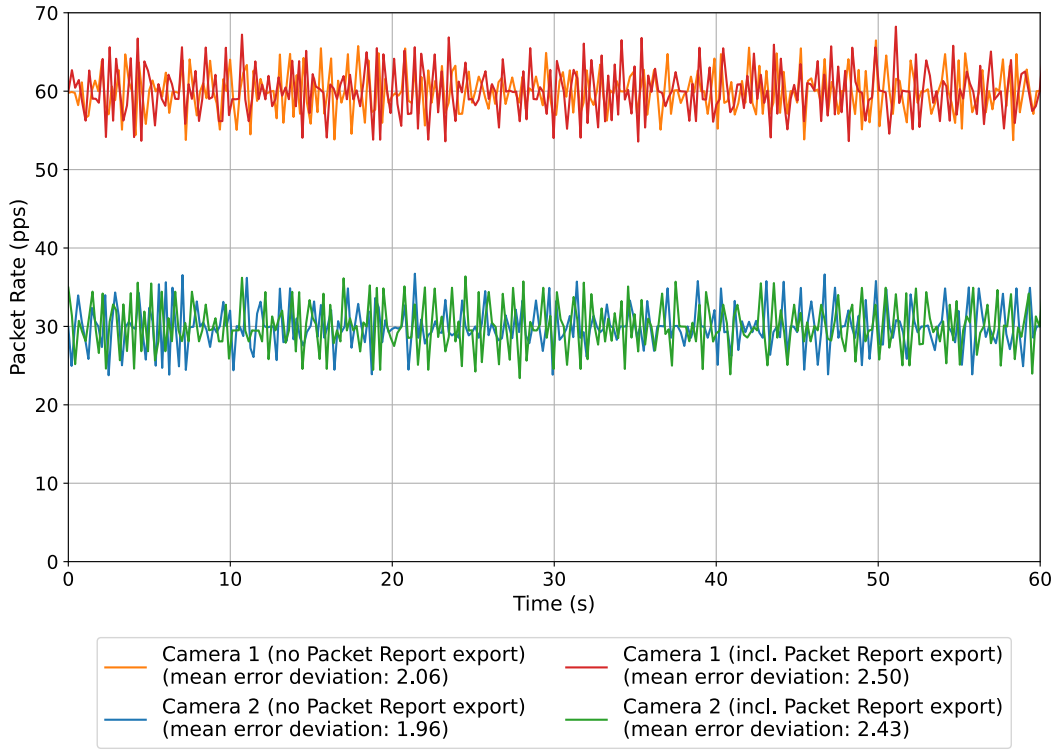

Figure 8.5: Python prototype: packet rate measurement comparison (Report Interpretation-based, export interval: 0.2 s), the inclusion of Packet Report export reduces the accuracy

Test Result 2: Figure 8.4 presents the results of a test run with an export interval of 0.1s. The fluctuations have increased compared to the 0.2s case, reaching an 18 pps difference from the average. The mean error deviation almost doubles to a rounded 3.9 pps for both Camera Flows. The increased granularity causes the calculation of the packet rate to be more sensitive to Report Interpretation transmission delay variations. The average packet rate remains similarly accurate as in the first test result.

Both presented tests leave the PSAMP Collector at a very low load of just 5 and 10 PSAMP messages per second. We expect the packet rate calculation to become even more inaccurate with increased load at the PSAMP Collector. For example, Packet Reports, if configured in our cases, add 90 extra PSAMP messages per second to the parsing pipeline within the PSAMP Collector.

Test Result 3: Figure 8.5 shows the results from Figure 8.3 combined with a second set of test results that represent the case of the inclusion of Packet Reports. Both test

results are similar, but looking at their mean error rate, it becomes clear that the test including Packet Reports exhibits increased fluctuations.

In general, this method of packet rate determination appears suitable for use cases where high precision is not necessary. As demonstrated, this approach becomes more accurate at lower granularity levels—however, increased load on the PSAMP Collectors parsing pipeline results in less precise measurement outcomes. To achieve precise metrics using Report Interpretations, introducing a new enterprise-specific Information Element, utilized as Report Interpretation timestamps, would be necessary.

Test Case 2 - Packet Report-based

Using the packet selection architecture, the second strategy to determine the packet rate of the two Camera Flows is based on Packet Reports.

Configuration: Each Packet Report is configured as shown in Listing 8.2. This configuration represents a Basic Packet Report [7] without the inclusion of any packet bytes. Using this approach allows the Network Health Monitor to calculate the exact packet rate as it receives Packet Reports of each packet belonging to a Camera Flow, including their respective timestamps. The precision of the timestamps depends on the PSAMP Device implementation. In our case, the `observationTimeNanoseconds` Information Element represents hardware packet timestamps measured at the NIC (Network Interface Card) of Test Node 2.

```
"basicPacketReport": {
    "observationDomainId": "0",
    "templateId": "258",
    "setId": "2",
    "field": [
        { "ieName": "selectionSequenceId", "ieLength": "4" },
        { "ieName": "observationTimeNanoseconds", "ieLength": "8" }
    ]
}
```

Listing 8.2: Basic Packet Report template configuration used to export packet timestamps

Test Result: Figure 8.6 presents the resulting measurements calculated at a resolution of 0.2 s. As anticipated, the Packet Report-based approach exhibits remarkable accuracy without any deviations, primarily due to using NIC hardware timestamps.

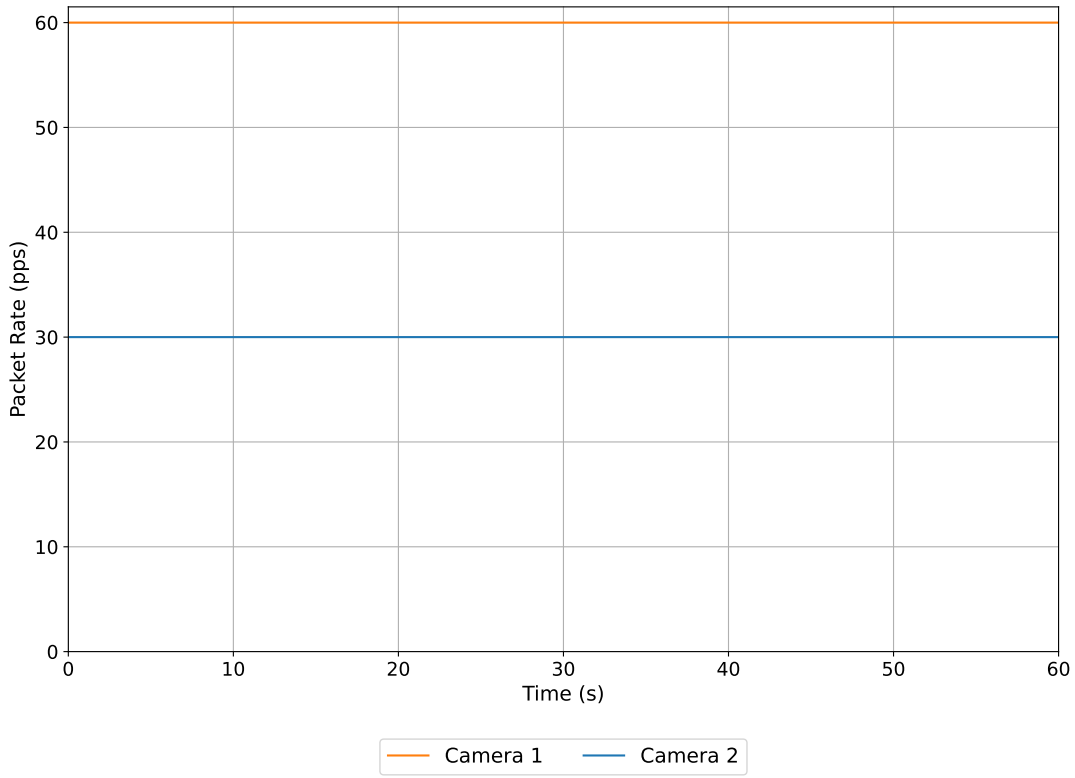

Figure 8.6: Python prototype: packet rate measurement (Packet Report-based, granularity: 0.2 s), the measurement is accurate

Conclusion

Comparing both packet rate measurement approaches, the Report Interpretation-based method exhibits lower accuracy, but its average measurements over more extended periods remain reliable. Using Report Interpretations packet rate measurements is sensible in situations where a Network Health Monitor does not receive any Packet Reports but still requires approximations of packet rates at a traffic Observation Point. This may be particularly relevant for scenarios in which Packet Report export is not feasible due to performance constraints of the Network Health Monitor or the PSAMP Device. By employing Packet Report Interpretations, a Network Health Monitor can monitor the packet rates of multiple Packet Flows at multiple Observation Points with minimal computing requirements, as low as parsing 5 pps per Observation Point. On the other hand, the Packet Report-based approach is optimized for maximum accuracy, which ultimately depends on the accuracy of packet timestamps. However, the performance requirements are generally higher and less predictable as the number of Packet Reports that a Network Health Monitor needs to parse is directly proportional to the packet rate of the filtered Packet Flow. Both approaches have their advantages. The choice between them depends on the accuracy requirements and feasibility of parsing Packet Reports.

8.2.2 Packet Interarrival Times: Python Prototype vs. Vermont

These test cases will compare the Vermont and Python prototype implementations of the PSAMP Device. Because Vermont can only filter for a single Packet Flow at a time, the Python variant will also be configured to filter for a single Packet Flow. The packet interarrival times can be calculated from the packet timestamps included in the Packet Reports as described previously in Chapter 6. Both test cases rely again on the Dual Camera test scenario. Both programs are configured to filter for Camera Flow 1.

Test Case 1 - Python Prototype

The Python prototype-based implementation offers PSAMP selection logic and support for high-precision timestamps reported to a nanosecond accuracy.

Configuration: The configuration of the Packet Report is identical to the Basic Packet Report used in the packet rate test. The Selection Process, filtering for the IP 5-tuple identifying Camera Flow 1, is configured as in Listing 8.3. The Selection Process consists of a composition of five property-matching Selectors, each filtering for a specific packet property. A packet is selected only if it matches all five Selectors.

```
"selectionProcess": [{
    "name": "SelectCameraFlow1",
    "selector": [
        {"name": "cameraDestinationPort", "filterMatch": true,
         "ieName": "destinationTransportPort", "value": "81"},
        {"name": "cameraSourcePort", "filterMatch": true,
         "ieName": "sourceTransportPort", "value": "8588"},
        {"name": "cameraUDPDestinationPort", "filterMatch": true,
         "ieName": "udpDestinationPort", "value": "81"},
        {"name": "cameraSourceIPv4Address", "filterMatch": true,
         "ieName": "sourceIPv4Address", "value": "192.168.2.37"},
        {"name": "cameraDestinationIPv4Address", "filterMatch": true,
         "ieName": "destinationIPv4Address", "value": "192.168.2.100"}
    ]
}]
```

Listing 8.3: Camera Flow 1 Selection Process configuration to select packets of Camera 1 by filtering for the IP 5-tuple

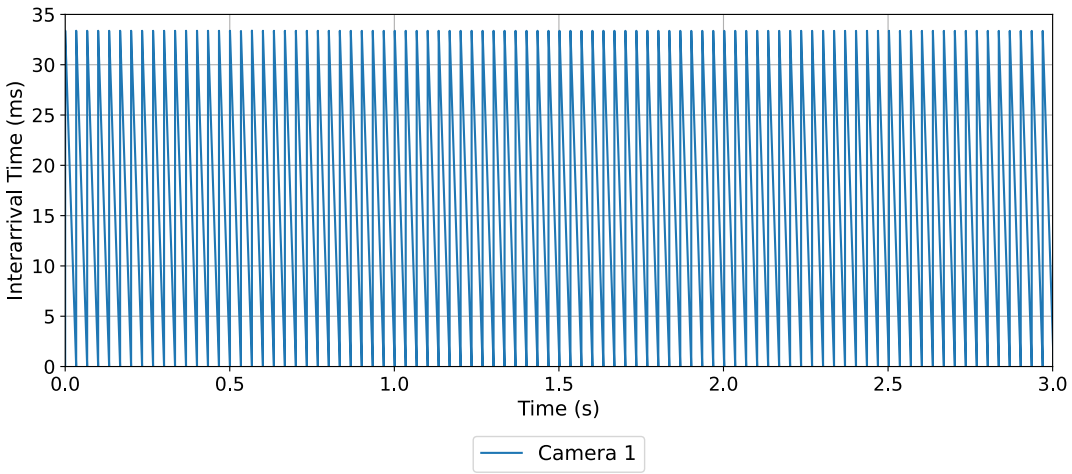

Figure 8.7: Python prototype: packet interarrival times from Packet Reports, the measurement is accurate

Test Result: Figure 8.7 displays the packet interarrival times between the packets in a 3-second snapshot of the test run. Evidently, the packet interarrival times consistently alternate between 33.3 ms, a thirtieth of a second, and 0 ms. This result perfectly aligns with the expectation displayed in Table 8.1.

Test Case 2 - Vermont

Vermont provides limited packet filtering logic in comparison to the Python prototype. Filtering for the IP 5-tuple of Packet Flows is not possible. However, in this test scenario, Camera Flow 1 has a unique source IP address and can be filtered using the IP address filter provided. This simple IP address filter significantly limits Vermont's functionality as a PSAMP Device within our packet selection architecture. It is also important to note that, when using Vermont, timestamps can only be reported with millisecond accuracy.

Configuration: The configuration of the Packet Filter component of Vermont is as in Listing 8.4. Additionally, the export of PSAMP Packet Reports is configured within the PSAMP Exporter module of Vermont, including the timestamp represented by the `observationTimeMilliseconds` Information Element. The Packet Report is configured as shown in Listing 8.5.

```
<filter id="3">
<hostBased>
    <addrFilter>src</addrFilter>
    <ip>192.168.2.37</ip>
</hostBased>
<next>4</next>
</filter>
```

Listing 8.4: Camera Flow 1 Vermont Packet Filter component configuration to filter for the source IP address of Camera 1

```
<packetReporting>
    <templateId>258</templateId>
    <reportedIE>
        <ieName>observationTimeMilliseconds</ieName>
    </reportedIE>
</packetReporting>
```

Listing 8.5: Vermont Basic Packet Report configuration for the export of packet timestamps

Test Result: Figure 8.8 displays the interarrival times between the packets in the same 3-second snapshot of the test run as before in the Python prototype test run. Analogous to Figure 8.7, we can observe an alternating pattern. However, we can not observe the

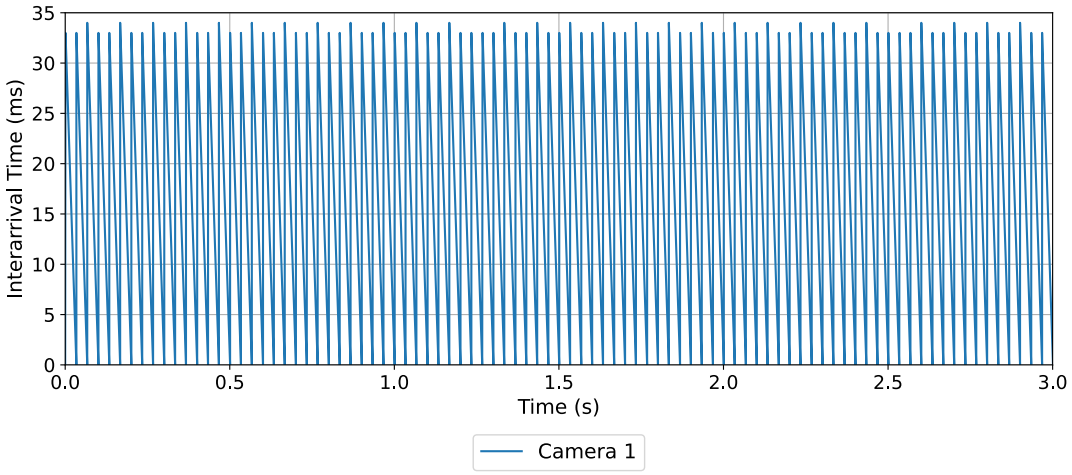

Figure 8.8: Vermont prototype: packet interarrival time measurement (Packet Report-based), the accuracy is limited to milliseconds by the utilized Information Element (Listing 8.5)

expected constant 33.3 ms delay between packet bursts in the Vermont test run. Instead, the interarrival time between the bursts follows the pattern: 33 ms, 33 ms, 34 ms. This pattern is a direct result of the limited millisecond accuracy of the exported timestamps. On average, the burst interarrival times seen in Figure 8.8 are the same as in Figure 8.7.

Conclusion

This experiment highlights the limitations in Vermont's packet classification and filtering, as only a single packet stream, identified by its source IP address, can be classified and filtered for. Furthermore, this filter would not be effective if there were any other Packet Flows with identical source IP addresses. Additionally, the accuracy of timestamps is limited to milliseconds, leading to less precise measurements. Our packet selection architecture implemented using Vermont does not fulfill the requirements for a packet selection architecture defined in Chapter 4 primarily due to the limited filtering capabilities.

8.2.3 Throughput

This test demonstrates the throughput measurement based on Packet Reports as discussed in Chapter 6. We will measure the throughput of Camera Flow 1 in the Dual Camera test scenario. Since Vermont does not support Information Elements reporting on packet length, only the Python prototype will be demonstrated.

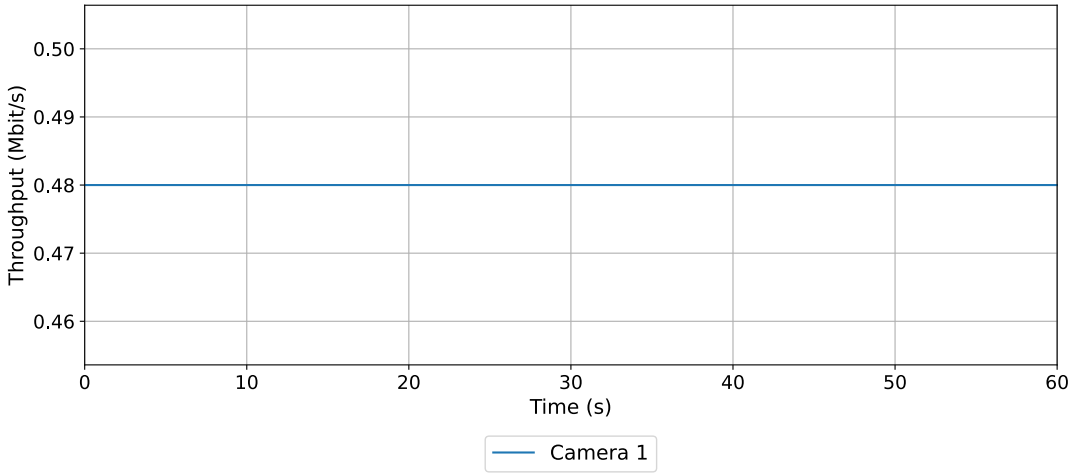

Figure 8.9: Python prototype: data throughput measurement (Packet Report-based, granularity: 0.2 s), the measurement is accurate

Configuration: The Python prototype is configured to export Packet Reports that include the `ethernetTotalLength` Information Element, as shown in Listing 8.6. This Information Element represents the total length of an Ethernet frame [9]. Following this

```
"throughputPacketReport": {
    "observationDomainId": "0", "templateId": "259", "setId": "2",
    "field": [
        {"ieName": "selectionSequenceId", "ieLength": "4"},
        {"ieName": "observationTimeNanoseconds", "ieLength": "8"},
        {"ieName": "ethernetTotalLength", "ieLength": "2"}
    ]}
```

Listing 8.6: Throughput Packet Report template configuration used to export packet lengths and packet timestamps

configuration, the PSAMP Device reports the `ethernetTotalLength` of every packet that belongs to Camera Flow 1, enabling the calculation of the Flow's throughput.

Test Result: As shown in Figure 8.9, the calculated throughput over time remains stable at 0.48 Mbit/s as anticipated in Table 8.1. This measurement stability is expected, as similar to the packet rate test, Packet Reports provide fine-grained per-packet information about the Camera Flow. This test, therefore, demonstrates that Packet Reports serve as a reliable foundation for measuring the throughput of Packet Flows.

	Camera 1	**Camera 2**	**Background Cameras 3-6**
Transport Protocol	UDP	UDP	UDP
Image Size (B)	1916	1458	1458
Ethernet Frames			
Frame 1 Size (B)	1500	1500	1500
Frame 2 Size (B)	500	-	-
Image Rate (fps)	30	30	100 - 1000 depending on Test Scenario

Table 8.2: Performance test scenarios: Dual Camera + background traffic, the inclusion of background traffic enables performance benchmarking of the PSAMP Device implementations

8.2.4 Performance Comparison: Python Prototype vs. Vermont

So far, the Python prototype has demonstrated functional superiority to Vermont without ever considering performance. All test cases have been conducted in the Dual Camera scenario at a low packet rate of 90 pps and without background traffic, which is suitable for demonstration purposes. However, real-life scenarios involve busy networks, including numerous Packet Flows.

Test Scenarios - Dual Cameras + Background Traffic

New test scenarios are required to compare the performance of both PSAMP Device implementations effectively. All new scenarios contain the same Camera Flows 1 and 2 as in the Dual Cameras test scenario. Additionally, background traffic is added to measure application performance at higher packet rates. The background traffic consists of four identical additional Camera Flows as displayed in Table 8.2. The five test scenarios with their respective configurations are listed in Table 8.3.

Test Scenario	Image Rate per Background Camera	Total Throughput	Total Packet Rate
TS 100	100 fps	5.64 Mbit/s	490 pps
TS 150	150 fps	8.04 Mbit/s	690 pps
TS 200	200 fps	10.44 Mbit/s	890 pps
TS 400	400 fps	20.04 Mbit/s	1690 pps
TS 1000	1000 fps	48.84 Mbit/s	4090 pps

Table 8.3: PSAMP Device Performance Test Scenarios

Configuration: For testing purposes, both implementations of a PSAMP Device will be configured identically to filter for the same source IP addresses identifying Camera Flow 1. The performance of the applications will be evaluated based on the quality of the Camera Flow 1 packet rate measurements with varying amounts of background traffic. It is anticipated that when the performance limit is reached, the PSAMP Device implementation will drop packets at the packet selection process and consequently only export a fraction of the Packet Reports compared to previous low-load scenarios. This packet-dropping behavior will be evident in Flow metrics such as the packet rate metric due to fewer Packet Reports being exported. The performance difference between both applications in the five scenarios can be deduced from this Packet Report loss. Each scenario is executed four times, and the test results are aggregated into an average result.

Performance Test Results

The Python prototype was not developed with a specific performance target in mind. Considering its short development time as part of this thesis and implementation based on Python, the performance is generally expected to be lower than Vermont, which was developed in C and C++ by a group of researchers at the Chair of Network Architectures and Services [2] at the Technical University of Munich.

Test Scenario	Python Average Packet Report loss	Vermont Average Packet Report loss
TS 100	<0.01 %	0 %
TS 150	6.15 %	0 %
TS 200	8.29 %	0 %
TS 400	28.41 %	0 %
TS 1000	66.96 %	0 %

Table 8.4: PSAMP Device Performance Test Results, Vermont is perfectly reliable throughout the tests, the Python prototype experiences varying degrees of Packet Report loss

Table 8.4 displays the average test results of both implementations. As observable, the Python prototype already exhibits considerable Packet Report loss of 6.15 % in the **TS 150** test scenario. The higher the background traffic, the more Packet Report loss can be observed. Only the **TS 100** test scenario exhibits nearly no Packet Report loss, with just a few packets dropped in one of the four test runs. Based on these performance measurements, it can be concluded that the performance limitation of the Python prototype lies at approximately 490 pps, the total packet rate of **TS 100**. If the Observed Packet Stream's packet rate increases, the implementation cannot handle the packet load, which leads to an unpredictable loss of Packet Reports. Consequently, the

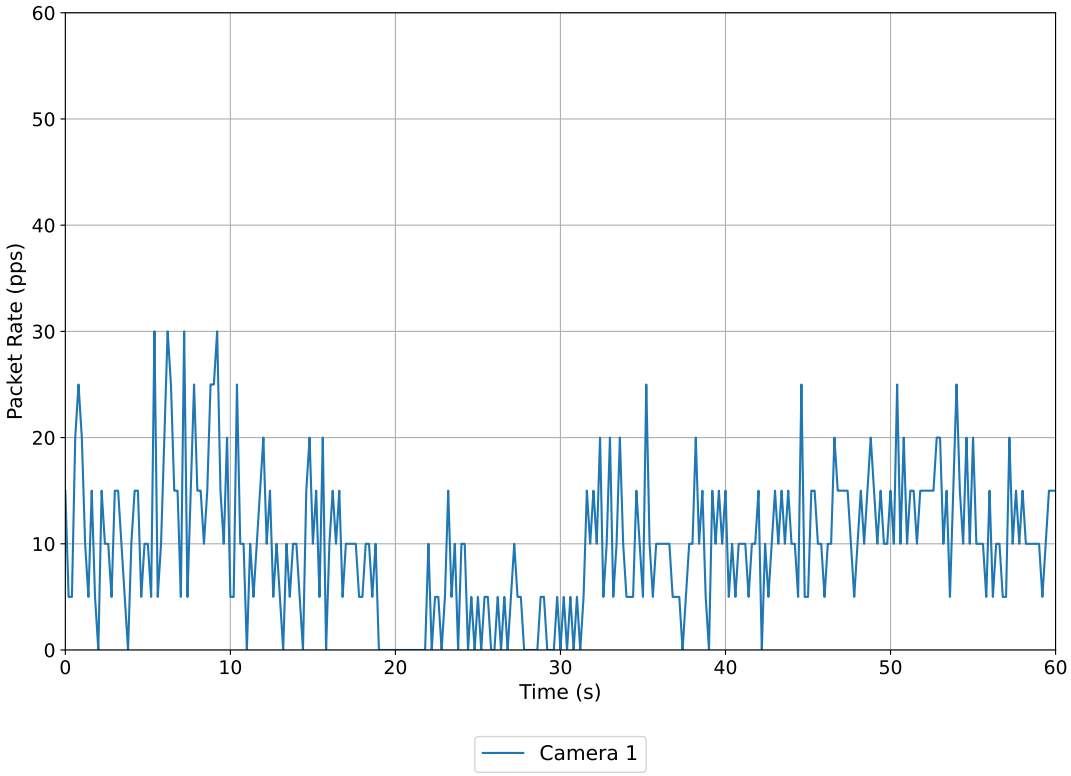

Figure 8.10: Python prototype: packet rate measurement (performance test scenario: **TS 1000**, Packet Report loss: 83.45 %), an example of critical Packet Report loss

Python prototype is unsuitable for scenarios with packet rates above 490 pps, as Packet Report loss results in information loss concerning the Packet Flow metrics.

Figure 8.10 shows an example of a packet rate measurement in the **TS 1000** test scenario with a high Packet Report loss rate of 83.45 %. Previously, without introducing background traffic, the packet rate remained constant at 60 pps, as seen in Figure 8.6. However, the **TS 1000** test run demonstrates how packets are lost throughout the experiment due to the increased packet rate. Observing multiple test runs, no specific pattern of Packet Report loss can be recognized.

In contrast, the test results for Vermont in Table 8.4 are flawless, as no Packet Report loss is observed in any of the tested scenarios.

Conclusion

Comparing the performance of the Python prototype to the Vermont implementation, it becomes clear that while the Python prototype does an adequate job of demonstrating the theoretical possibilities of the packet selection architecture, performance was

not a target during its development. Consequently, the Python prototype is currently unsuitable for deployment in realistic test scenarios at packet rates exceeding 490 pps. Vermont does not exhibit any performance constraints in the tested scenarios at packet rates up to 4090 pps, thus being significantly more efficient than the Python prototype. This performance result is very promising and should be a reason for expanding the functions of Vermont to support all aspects of our packet selection architecture.

8.3 Tetrys Architecture Concept Test Scenario

This section aims to conceptualize a test scenario for the Tetrys Architecture Concept introduced in Chapter 6 and describe the expected behavior of the various components during the test scenario. However, there is currently no implementation available to test the scenario. In the Tetrys use case, the packet selection architecture serves a significantly different purpose than in the Network Health Monitoring case. It aids the Tetrys Network Coding protocol in ensuring the reliability of specific Packet Flows. This is accomplished through a distributed Tetrys network function partly embedded in PSAMP Collector, a PSAMP Device, and the control plane, as illustrated in Figure 6.2.

8.3.1 Test Scenario Definition

The Dual Camera test scenario from section 8.1 serves as a basis for evaluating the behavior of the Tetrys concept architecture. However, we slightly modify the scenario by introducing dynamic packet loss. This should adequately demonstrate the Tetrys Architecture behavior designed to handle dynamic network conditions.

Both Camera Flows of the test scenario are known to the control plane component, with Camera Flow 1 having a higher reliability requirement than Camera Flow 2. This difference in reliability requirements leads to distinct calculated redundancy ratios for Camera Flows 1 and 2.

The PSAMP Device, configured by the control plane, has two Selection Processes, each filtering for one Camera Flow using the respective IP 5-tuple. The Exporting Process of the PSAMP Device is configured to export Packet Reports containing `flowId`, `dataLinkFrameSection`, and `redundancyRatio` values, as defined in Chapter 6. To generate these Packet Reports, the PSAMP Device maintains an internal table containing `flowIds` and their respective `redundancyRatios`. The control plane regularly updates this table with redundancy ratios calculated from the packet loss monitored by Tetrys and the Flow redundancy requirements.

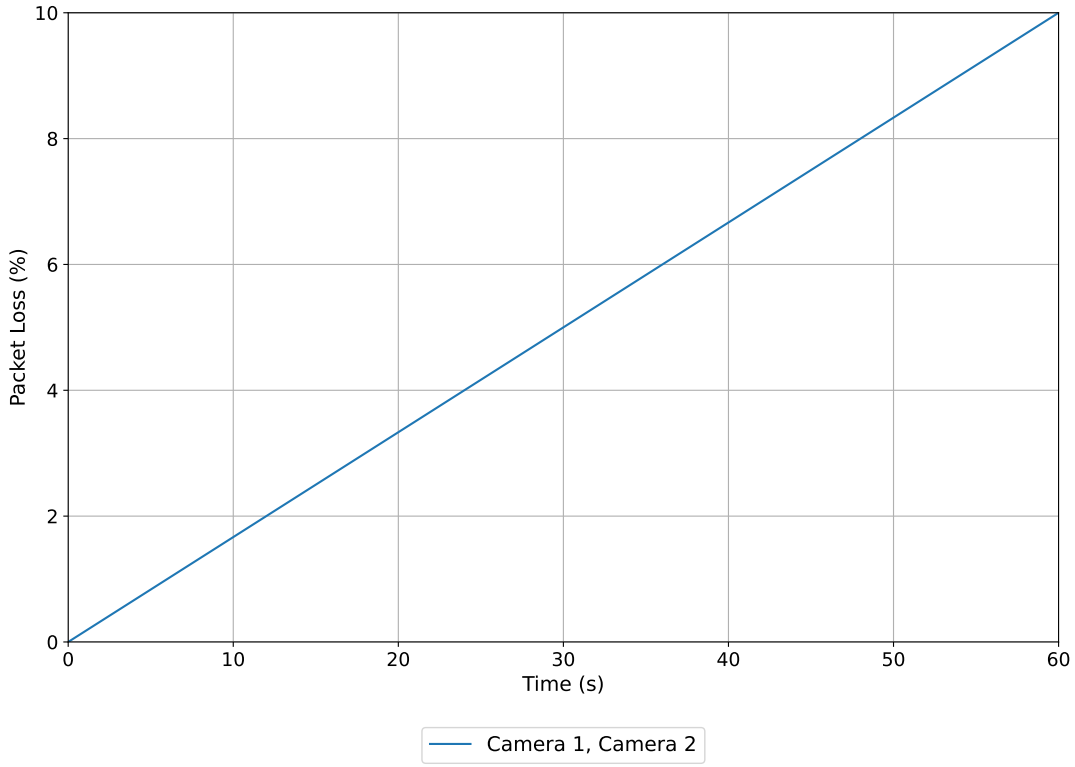

Figure 8.11: Tetrys Architecture Concept: packet loss scenario, the packet loss increases linearly over time, the Tetrys Architecture is expected to react by increasing the coding redundancy proportionally to the packet loss

As described in Chapter 6, the Tetrys Encoder generates coded packets using the packet captures and the per-Flow redundancy ratio from the Packet Reports. The coded packets are transmitted to the Tetrys Decoder, where they are decoded and reinserted into the network as regular packets without Tetrys encapsulation. Tetrys reports any experienced packet loss to the control plane throughout the transmission. The inner workings of the Tetrys network function and the control plane are treated as black boxes and not discussed in further detail.

8.3.2 Tetrys Architecture Test Behaviour

In this test scenario, the key performance indicators of the Tetrys Architecture are the redundancy ratio adjustments by the control plane and the overall success rate of the architecture in preventing packet loss. By monitoring these indicators, we can assess whether the architecture effectively responds to changes in packet loss.

For instance, when considering a linearly increasing packet loss as depicted in Figure 8.11, we expect the control plane to react accordingly by proportionally increasing

the redundancy ratios for both Camera Flows in response to the packet loss. Due to the higher reliability requirement of Camera Flow 1, its redundancy ratio is always expected to remain higher than that of Camera Flow 2. Ideally, the correct delivery of the Camera Flow packets at the destination node B is consistent for both Flows. Following its requirement, Camera Flow 1 is generally expected to be more reliable than Camera Flow 2.

This hypothetical test scenario provides insights into the Tetrys Architecture's potential ability to dynamically adapt redundancy ratios for different Flows, responding to changing network conditions in the form of packet loss. By successfully delivering packets at the destination node, the architecture demonstrates its effectiveness in handling loss-sensitive Camera Flows used for computer vision applications within a lossy network environment. Furthermore, this test scenario serves as a template for evaluating future implementations of the Tetrys Architecture Concept.

8.4 Flexibility Evaluation of the Architecture

One of the primary motivations for developing the new packet selection architecture was to achieve flexibility toward a wide range of network functions. This goal was pursued and questioned throughout the thesis. This section provides a final analysis, including the experience from implementation and testing.

Overall, the conceptualized packet selection architecture is flexible regarding packet selection and selection results, which can be configured to match network function requirements. The critical factor in achieving this flexibility was the incorporation of IPFIX Information Elements, which are already multifaceted in their current form as defined in the IANA registry [9] but can also be extended by utilizing enterprise-specific Information Elements. The packet selection mechanism and the selection results are based on these Information Elements. Consequently, packet filters can be configured to filter for a diverse set of packet properties, header fields, and derived packet properties. Furthermore, combining various packet filters and packet sampling facilitates packet selection flexibility. The composition of the selection results is variable. The Information Elements included are defined by templates configurable depending on network function requirements. Furthermore, an alternate information source for network functions is provided as Report Interpretations, also consisting of Information Elements and flexibly defined by templates. In contrast to Packet Reports, the Report Interpretations are not direct selection results but originate from statistics, configuration, and operational data collected during the operation of the packet selection exported at regular intervals.

The clear architectural composition of the architecture, which includes a PSAMP Device for packet observation, selection, and exporting, as well as a PSAMP Collector for packet result collection and network function, presents a modular and component-based approach. This modularity facilitates the development of more complex network function architectures with distributed components, potentially including multiple PSAMP Devices and PSAMP Collectors, enabling the implementation of distributed network functions such as network monitors. This modularity also ensures compatibility with other IPFIX-related standards not yet integrated, like Mediators as described in [45]. Mediators provide a framework for middleboxes located between the PSAMP Device and PSAMP Collector. As demonstrated in [15], they can help overcome performance constraints and improve the scalability of network functions.

8.4.1 Flexibility Limitations

The flexibility of the architecture is limited for some particular use cases. Also, the inclusion of certain concepts was neglected in favor of the maximum generality of the approach.

As previously mentioned in Chapter 5, Flow aggregation and the export of Flow Records were excluded from the architecture to ensure a more general architecture. However, there is also a downside to this choice. For Flow-based network functions that require Flow aggregation, this ultimately leads to an offloading of Flow aggregation from the architecture to the network function resulting in increased network function complexity. The choice against Flow Records was made because Flow aggregation vastly limits the generality of the architecture, even though its exclusion contradicts the goal of simplifying network function implementations.

After testing our architecture in various Network Health Monitoring test scenarios, we can observe the impact of incorporating Packet Reports. For instance, all Flow measurements showcased in the implementation tests, e.g., packet rate metrics over time, would not be possible using Flow aggregation, as that concept does not allow gathering fine-grained statistics about Flow behavior. Instead, only one Flow Record is exported at the end of a Packet Flow, containing aggregated information resulting from all the Flow's packets. Although the Flow's packet rate could be derived from metrics in the Flow Records, such as packet counts and timestamps at the beginning and end of the Flow, this only results in an average packet rate value. In contrast, our packet selection architecture can provide the exact packet rates at various points in time throughout the Flow's existence.

The trade-off is between Flow Records, a highly efficient solution providing limited information, and Packet Reports, a resource-intensive solution providing detailed statistics about packets and Packet Flows. The decision between the two depends on the specific use case. However, Packet Reports are generally more flexible. Therefore the choice of Packet Reports for this architecture is reasonable.

Another limitation of our architecture, which originates from its reliance on PSAMP Selection Processes, is that certain packet selection logic can not be expressed. This issue became evident during the analysis of potential network functions to implement. Firewall rule sets, for example, in PF, are often represented by an ordered list of filter rules [18]. In PF, these rule lists are evaluated sequentially, and the last rule to match determines the action taken on the packet. This property of the firewall filter rules can not be translated to the parallel concept of PSAMP Selection Processes. Consequently, the packet selection architecture may not be suitable for supporting firewall network functions in its current form. Before applying the architecture to support a network function, analyzing whether the desired selection logic can be expressed using PSAMP Selection Processes is crucial.

8.5 Summary

Various test scenarios demonstrated the packet selection architecture's capabilities in supporting a Network Health Monitor. These tests focused on monitoring Packet Flows originating from Cameras used for a computer vision application. In particular, packet rate, packet interarrival times, and network throughput were measured. The Python prototype proved to be an effective, feature-complete implementation of the packet selection architecture, capable of simultaneously gathering various metrics on multiple Packet Flows. While lacking vital architectural features such as complex packet filters, certain Information Elements, and the export of Report Interpretations, Vermont is still suitable for specific scenarios, such as measuring the packet interarrival times of a single Packet Flow. In the performance test, Vermont demonstrated excellent reliability at packet rates up to 4090 pps, while the Python prototype operated reliably at up to 490 pps. In a hypothetical test scenario, the Tetrys Architecture Concept showcased its potential for ensuring reliable delivery of image data from cameras despite a lossy network. The architecture's reliance on PSAMP limits selection logic in specific edge-case applications, such as firewalls. However, generally speaking, the IPFIX and PSAMP concepts achieve great flexibility in the packet selection architecture.

In conclusion, this chapter successfully demonstrated and validated the functionality and flexibility of our packet selection architecture using open-source software and a custom Python implementation.

Chapter 9

Conclusion

In this final chapter, we reflect on our research's key takeaways. We will also discuss how our architecture relates to existing research and outline potential future work in this area.

9.1 Summary of results

This thesis aimed to identify relevant IPFIX/PSAMP components and data models to develop a new, flexible packet selection architecture. Based on the defined architecture requirements, we discovered that the PSAMP framework includes all the necessary components and mechanisms. PSAMP is preferred over IPFIX as the foundation for the architecture because it addresses the `Low-Latency Processing` and `Full Capture Provision` limitations of IPFIX, as discussed in Chapter 4. In the final architecture, packets are selected within the PSAMP Device and exported as Packet Reports to the PSAMP Collector, which then forwards the results to the network function. We chose to utilize Packet Reports in favor of Flow Records, eliminating Flow aggregation to maximize the flexibility in the resulting architecture.

The proposed packet selection architecture provides several advantages and opportunities for network functions. These were demonstrated with the case-study examples, Network Health Monitoring, and the Network Coding protocol Tetrys. The new architecture provides Network Health Monitors with a standardized framework to monitor network traffic simply and efficiently in a distributed manner, including the measurements of various traffic properties. The architecture also enables an optional inclusion of packet sampling, which can be combined with packet filters. The functionality of a Network Health Monitor implemented using the packet selection architecture is validated

by testing the implementation prototypes. For the Tetrys Network Coding protocol, the architecture presents a versatile solution that combines its architecture and Tetrys components. In the Tetrys Architecture Concept, this combination achieves dynamic transmission safety for a network stream of loss-sensitive Flows by monitoring packet loss and dynamically adjusting coding redundancy for each Flow.

The packet selection architecture successfully meets its objective of supporting various types of network functions through its packet selection and selection result export mechanisms. The Python Network Health Monitoring implementation demonstrated the simplicity of incorporating network functions within the proposed architecture. The Tetrys Architecture Concept emphasizes the benefits of compliance with internet standards, enabling seamless integration into complex network architectures. Overall, the architecture achieves its main objectives.

9.2 Future Work

Due to the limited scope of this thesis, the architecture was validated by analyzing two possible use cases. However, to comprehensively assess the architecture's applicability, additional network functions should be evaluated. These assessments should also consider Hash-based Selection, another packet selection mechanism besides packet filtering. By doing so, it can be demonstrated that there is no specific relation between the final architecture model and the chosen use cases. Investigating use cases that IPFIX/PSAMP were not initially designed for, such as the Tetrys Architecture Concept, would reveal new applications and validate the architecture's versatility.

For such extended research, a high-performing architecture implementation is necessary. This implementation can be based on Vermont and incorporate missing functional components, including the support for more Information Elements, the export of Report Interpretations, and PSAMP packet selection. With this high-performing implementation, more practical experiments and applications of the packet selection architecture can be conducted.

Moreover, a high-performing implementation would enable research on the architecture's effectiveness in high-performance environments. When utilizing the architecture for large-scale applications, potential extensions to the architecture could be considered, such as incorporating other IPFIX-based standards like IPFIX Mediators [45].

Lastly, further research could focus on improving the architecture's security and resilience, addressing potential vulnerabilities and threats in the context of real-world deployments. This would involve reviewing the security considerations outlined in the

various RFC documents that the architecture is based on. By doing so, researchers can identify potential weak points and areas for improvement. This could involve integrating security measures into the architecture, such as encryption and authentication, to ensure the confidentiality, integrity, and availability of the packet selection process and its results.

Chapter A

Appendix

A.1 List of Acronyms

BPF Berkeley Packet Filter

IANA Internet Assigned Numbers Authority

IAT Interarrival Times

IE Information Element

IETF Internet Engineering Task Force

IPFIX IP Flow Information Export

JSON JavaScript Object Notation

NETCONF Network Configuration Protocol

NIC Network Interface Controller

NTP Network Time Protocol

PCAP Packet Capture

POS Plain Orchestrating Service

PSAMP Packet Sampling

QoS Quality of Service

RFC Request for Comments

TCP Transmission Control Protocol

UDP User Datagram Protocol

XML Extensible Markup Language

YANG Yet Another Next Generation

Bibliography

[1] T. Zseby, E. Boschi, N. Brownlee, and B. Claise, *IP Flow Information Export (IPFIX) Applicability*, RFC 5472, Mar. 2009. DOI: 10.17487/RFC5472. [Online]. Available: https://www.rfc-editor.org/info/rfc5472.

[2] *Vermont*, https://github.com/tumi8/vermont/wiki, Accessed: 2023-04-18.

[3] B. Claise and B. Trammell, *Information Model for IP Flow Information Export (IPFIX)*, RFC 7012, Sep. 2013. DOI: 10.17487/RFC7012. [Online]. Available: https://www.rfc-editor.org/info/rfc7012.

[4] P. Aitken, B. Claise, and B. Trammell, *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information*, RFC 7011, Sep. 2013. DOI: 10.17487/RFC7011. [Online]. Available: https://www.rfc-editor.org/info/rfc7011.

[5] D. N. Duffield, M. Molina, F. Raspall, S. Niccolini, and T. Zseby, *Sampling and Filtering Techniques for IP Packet Selection*, RFC 5475, Mar. 2009. DOI: 10.17487/RFC5475. [Online]. Available: https://www.rfc-editor.org/info/rfc5475.

[6] J. Rexford, D. Chiou, M. Grossglauser, B. Claise, A. Greenberg, and D. N. Duffield, *A Framework for Packet Selection and Reporting*, RFC 5474, Mar. 2009. DOI: 10.17487/RFC5474. [Online]. Available: https://www.rfc-editor.org/info/rfc5474.

[7] A. Johnson, J. Quittek, and B. Claise, *Packet Sampling (PSAMP) Protocol Specifications*, RFC 5476, Mar. 2009. DOI: 10.17487/RFC5476. [Online]. Available: https://www.rfc-editor.org/info/rfc5476.

[8] G. Sadasivan, N. Brownlee, B. Claise, and J. Quittek, *Architecture for IP Flow Information Export*, RFC 5470, Mar. 2009. DOI: 10.17487/RFC5470. [Online]. Available: https://www.rfc-editor.org/info/rfc5470.

[9] *IP Flow Information Export (IPFIX) Entities*, https://www.iana.org/assignments/ipfix/ipfix.xhtml, Accessed: 2023-04-07.

[10] B. Claise, T. Dietz, F. Dressler, G. Carle, and P. Aitken, *Information Model for Packet Sampling Exports*, RFC 5477, Mar. 2009. DOI: 10.17487/RFC5477. [Online]. Available: https://www.rfc-editor.org/info/rfc5477.

[11] *Tetrys, an On-the-Fly Network Coding Protocol*, https://www.ietf.org/archive/id/draft-irtf-nwcrg-tetrys-04.html, Accessed: 2023-12-03.

[12] F. Fatemipour and M. H. Yaghmaee, "Design and Implementation of a Monitoring System Based on IPFIX Protocol", in *The Third Advanced International Conference on Telecommunications (AICT'07)*, 2007, pp. 22–22. DOI: 10.1109/AICT.2007.18.

[13] G. Munz and G. Carle, "Distributed Network Analysis Using TOPAS and Wireshark", in *NOMS Workshops 2008 - IEEE Network Operations and Management Symposium Workshops*, 2008, pp. 161–164. DOI: 10.1109/NOMSW.2007.27.

[14] K. Kaneko, "Precise network monitoring for high quality live streaming", in *2011 Third International Conference on Ubiquitous and Future Networks (ICUFN)*, 2011, pp. 83–88. DOI: 10.1109/ICUFN.2011.5949140.

[15] A. Kobayashi, Y. Hirokawa, and H. Nishida, "IP Multicast Traffic Measurement Method with IPFIX/PSAMP", Sep. 2008, pp. 78–90, ISBN: 978-3-540-87356-3. DOI: 10.1007/978-3-540-87357-0_7.

[16] A. Neumann, M. Ehrlich, L. Wisniewski, and J. Jasperneite, "Towards monitoring of hybrid industrial networks", in *2017 IEEE 13th International Workshop on Factory Communication Systems (WFCS)*, 2017, pp. 1–4. DOI: 10.1109/WFCS.2017.7991973.

[17] R. Hofstede, V. Bartoš, A. Sperotto, and A. Pras, "Towards real-time intrusion detection for NetFlow and IPFIX", in *Proceedings of the 9th International Conference on Network and Service Management (CNSM 2013)*, 2013, pp. 227–234. DOI: 10.1109/CNSM.2013.6727841.

[18] *OpenBSD PF - User's Guide*, https://www.openbsd.org/faq/pf/, Accessed: 2023-04-07.

[19] "IEEE Standard for Information Technology–Portable Operating System Interface (POSIX(TM)) Base Specifications, Issue 7", *IEEE Std 1003.1-2017 (Revision of IEEE Std 1003.1-2008)*, pp. 1–3951, 2018. DOI: 10.1109/IEEESTD.2018.8277153.

[20] *nftables wiki*, https://wiki.nftables.org/, Accessed: 2023-04-07.

[21] N. Islam, C. C. Bawn, J. Hasan, A. I. Swapna, and M. S. Rahman, "Quality of service analysis of Ethernet network based on packet size", *Journal of Computer and communications*, vol. 4, no. 4, pp. 63–72, 2016.

[22] J. Iyengar and M. Thomson, *QUIC: A UDP-Based Multiplexed and Secure Transport*, RFC 9000, May 2021. DOI: 10.17487/RFC9000. [Online]. Available: https://www.rfc-editor.org/info/rfc9000.

[23] K. Holzinger, F. Biersack, H. Stubbe, *et al.*, "SmartNIC-based Load Management and Network Health Monitoring for Time Sensitive Applications", in *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*, 2022, pp. 1–6. DOI: 10.1109/NOMS54207.2022.9789863.

[24] *OpenBSD manual page server - pf.conf*, https://man.openbsd.org/pf.conf, Accessed: 2023-04-07.

[25] G. Muenz, B. Claise, and P. Aitken, *Configuration Data Model for the IP Flow Information Export (IPFIX) and Packet Sampling (PSAMP) Protocols*, RFC 6728, Oct. 2012. DOI: 10.17487/RFC6728. [Online]. Available: https://www.rfc-editor.org/info/rfc6728.

[26] G. Almes, S. Kalidindi, M. J. Zekauskas, and A. Morton, *A One-Way Delay Metric for IP Performance Metrics (IPPM)*, RFC 7679, Jan. 2016. DOI: 10.17487/RFC7679. [Online]. Available: https://www.rfc-editor.org/info/rfc7679.

[27] C. M. Demichelis and P. Chimento, *IP Packet Delay Variation Metric for IP Performance Metrics (IPPM)*, RFC 3393, Nov. 2002. DOI: 10.17487/RFC3393. [Online]. Available: https://www.rfc-editor.org/info/rfc3393.

[28] G. Almes, S. Kalidindi, M. J. Zekauskas, and A. Morton, *A One-Way Loss Metric for IP Performance Metrics (IPPM)*, RFC 7680, Jan. 2016. DOI: 10.17487/RFC7680. [Online]. Available: https://www.rfc-editor.org/info/rfc7680.

[29] R. Hofstede, P. Čeleda, B. Trammell, *et al.*, "Flow Monitoring Explained: From Packet Capture to Data Analysis With NetFlow and IPFIX", *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 2037–2064, 2014. DOI: 10.1109/COMST.2014.2321898.

[30] *nProbe documentation*, https://www.ntop.org/guides/nprobe/, Accessed: 2023-04-18.

[31] *pmacct wiki*, https://github.com/pmacct/pmacct/wiki, Accessed: 2023-04-18.

[32] *QoF*, https://github.com/britram/qof/, Accessed: 2023-04-18.

[33] *YAF: Documentation*, https://tools.netsa.cert.org/yaf/docs.html, Accessed: 2023-04-18.

[34] M. Bjorklund, *pyang*, https://github.com/mbj4668/pyang, Accessed: May 5, 2023.

[35] P. S. Foundation, *The Python Standard Library*, https://docs.python.org/3/library/index.html, Accessed: May 5, 2023.

[36] M. Blech, *xmltodict*, https://github.com/martinblech/xmltodict, Accessed: May 5, 2023.

[37] J. Martin, J. Burbank, W. Kasch, and P. D. L. Mills, *Network Time Protocol Version 4: Protocol and Algorithms Specification*, RFC 5905, Jun. 2010. DOI: 10.

17487/RFC5905. [Online]. Available: https://www.rfc-editor.org/info/rfc5905.

[38] *Scapy API reference scapy.layers.l2*, https://scapy.readthedocs.io/en/latest/api/scapy.layers.l2.html, Accessed: 2023-04-18.

[39] SecDev, *Scapy*, https://github.com/secdev/scapy, Accessed: May 5, 2023.

[40] L. Mark, M. Stiemerling, P. Aitken, J. Quittek, and E. Boschi, *IP Flow Information Export (IPFIX) Implementation Guidelines*, RFC 5153, Apr. 2008. DOI: 10.17487/RFC5153. [Online]. Available: https://www.rfc-editor.org/info/rfc5153.

[41] *Tcpreplay - Pcap editing and replaying utilities*, https://tcpreplay.appneta.com, Accessed: 2023-05-10.

[42] K. Holzinger, *IVN Traffic Scenarios*, https://gitlab.lrz.de/ivntig/scenarios, Accessed: May 5, 2023.

[43] TUM Chair for Network Architectures and Services, *POS Command Line Interface*, https://gitlab.lrz.de/I8-testbeds/pos/cli, Accessed: May 1, 2023, 2023.

[44] T. Doering, *PSAMP Packet Selection Architecture*, https://gitlab.lrz.de/netintum/teaching/tumi8-theses/ba-doering, Accessed: May 1, 2023, 2023.

[45] B. Claise, A. Kobayashi, G. Muenz, and K. Ishibashi, *IP Flow Information Export (IPFIX) Mediation: Framework*, RFC 6183, Apr. 2011. DOI: 10.17487/RFC6183. [Online]. Available: https://www.rfc-editor.org/info/rfc6183.