



BRNO UNIVERSITY OF TECHNOLOGY

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS

ÚSTAV INFORMAČNÍCH SYSTÉMŮ

PCAP NETFLOW V5 EXPORTER

PCAP NETFLOW V5 EXPORTÉR

TERM PROJECT

SEMESTRÁLNÍ PROJEKT

AUTHOR

AUTOR PRÁCE

MALASHCHUK VLADYSLAV

SUPERVISOR

VEDOUCÍ PRÁCE

Ing. KAMIL JEŘÁBEK, Ph.D.

BRNO 2024

Abstract

The project focuses on the development of a PCAP-based NetFlow v5 exporter, designed for Unix systems. This tool processes packet capture (PCAP) files, extracts TCP flow records, and aggregates them. The aggregated flow data is then exported to a specified NetFlow collector using the UDP protocol. The tool offers configurable options, such as active and inactive timeout parameters, allowing for flexibility in flow export timing. This solution is intended for network traffic analysis, security monitoring, and optimization of network performance.

Abstrakt

Projekt se zaměřuje na vývoj exportéru NetFlow v5 založeného na PCAP pro unixové systémy. Tento nástroj zpracovává soubory zachycení paketů (PCAP), extrahuje a agreguje záznamy TCP toků. Agregovaná data toků jsou následně exportována do specifikovaného NetFlow kolektoru pomocí protokolu UDP. Nástroj nabízí konfigurovatelné možnosti, jako je časový limit aktivních a neaktivních toků, což umožňuje flexibilitu při exportu toků. Toto řešení je určeno pro analýzu síťového provozu, monitorování bezpečnosti a optimalizaci výkonu sítě.

Keywords

PCAP, NetFlow v5, TCP flow records, Unix systems, network traffic analysis, UDP export, flow aggregation, network security.

Klíčová slova

PCAP, NetFlow v5, TCP záznamy toků, unixové systémy, analýza síťového provozu, export UDP, agregace toků, bezpečnost sítě.

Reference

VLADYSLAV, Malashchuk. *PCAP NetFlow v5 exporter*. Brno, 2024. Term project. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Kamil Jeřábek, Ph.D.

PCAP NetFlow v5 exporter

Declaration

Prohlašuji, že jsem tento projekt vypracoval samostatně. Další informace mi poskytl Internet. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Malashchuk Vladyslav
September 29, 2024

Acknowledgements

Thank me. And probably thanks to whoever invented the Internet, since all the information taken for the project was from there.

Contents

1	Introduction	3
1.1	Overview	3
1.2	Motivation	3
1.3	Scope	4
2	Review of Literature	5
2.1	NetFlow Protocol	5
2.2	PCAP File Structure	5
2.2.1	PCAP Parsing in C/C++	6
2.3	Packet Capture and Traffic Analysis Tools	6
2.3.1	Wireshark	6
2.3.2	tcpdump	6
2.3.3	nfdump and nfsen	7
2.4	Summary	7
3	Application Design	8
3.1	System Architecture	8
3.1.1	PCAP File Parsing	8
3.1.2	Flow Aggregation	8
3.1.3	NetFlow Exporting	9
3.2	Data Flow	9
3.3	Component Breakdown	9
3.3.1	PCAP File Reader	10
3.3.2	Flow Aggregator	10
3.3.3	NetFlow Exporter	10
3.3.4	Configuration Manager	11
3.4	Summary	11
4	Implementation	12
4.1	Language and Libraries	12
4.2	Interesting Implementation Details	12
4.2.1	Flow Aggregation Algorithm	12
4.2.2	Timeout Handling for Active and Inactive Flows	12
4.3	Program Structure	13
5	Program Usage	14
5.1	Installation	14
5.1.1	Dependencies	14

5.1.2	Installation Steps	14
5.2	Command Line Interface (CLI)	14
5.2.1	Running the Program	14
5.2.2	Parameter Description	15
5.2.3	Example Command	15
5.3	Usage Examples	15
5.3.1	Example 1: Basic Processing	15
5.3.2	Example 2: Setting Active Timeout	15
5.3.3	Example 3: Setting Inactive Timeout	15
5.3.4	Example 4: Using with DNS	16
5.3.5	Example 5: Command Help	16
6	Testing and Results	17
6.1	Test Setup	17
6.2	Test Cases	17
6.3	Results	17
6.4	Visual Results	18
7	Conclusion	20
8	Literature	21

Chapter 1

Introduction

1.1 Overview

The continuous growth of computer networks and the increasing demand for reliable data transmission have made network traffic monitoring and analysis essential for maintaining performance, detecting anomalies, and ensuring security. NetFlow, a network protocol developed by Cisco Systems, plays a key role in providing insights into network traffic by collecting flow data, which represents a sequence of packets between two network endpoints. The data is then analyzed to understand traffic patterns, optimize network usage, and detect any malicious activity.

This project is dedicated to the creation of a PCAP NetFlow v5 exporter, a specialized tool that processes Packet Capture (PCAP) files, extracts flow information from TCP traffic, and exports the data to a NetFlow collector. The focus of the exporter is on providing an efficient, accurate, and configurable means of aggregating network traffic flows. With its lightweight implementation and flexibility in handling PCAP inputs, this tool is designed for Unix systems where detailed traffic monitoring is required, making it valuable for both network administrators and security professionals.

1.2 Motivation

In today's digital environment, real-time monitoring and historical analysis of network traffic have become critical for troubleshooting, network performance optimization, and threat detection. While many tools can monitor live network traffic, capturing and analyzing previously recorded traffic in PCAP format provides a unique opportunity for retrospective analysis and forensic investigation. However, turning PCAP data into NetFlow records requires specialized software capable of processing packet data, identifying flows, and exporting them efficiently.

The motivation behind this project stems from the need to bridge the gap between raw packet data stored in PCAP files and the detailed insights provided by NetFlow v5 format, enabling seamless analysis in environments with varying security and performance requirements. By providing an open-source, easily configurable tool, this project aims to simplify network traffic analysis for system administrators and cybersecurity experts.

1.3 Scope

The PCAP NetFlow v5 exporter is designed for Unix-based systems and focuses on TCP flows. It reads PCAP files, aggregates flow information (such as source/destination IP addresses, ports, packet counts, and timestamps), and exports the data using the NetFlow v5 protocol via UDP to a collector. The tool also offers configurable parameters such as active and inactive timeouts to control flow export frequency.

The tool's target users include:

- **Network Administrators** who need insights into traffic patterns for performance monitoring.
- **Security Analysts** looking to analyze historical traffic for potential threats.
- **Researchers** investigating network behaviors based on past network activities.

Chapter 2

Review of Literature

This chapter summarizes relevant information from existing research, documentation, and technical papers that influenced the design and implementation of this project. The areas covered include the NetFlow protocol, PCAP file structure, and common packet capture and traffic analysis tools.

2.1 NetFlow Protocol

NetFlow is a network protocol developed by Cisco Systems in the 1990s. It allows for the collection, aggregation, and analysis of IP traffic information, giving network administrators the ability to gain insights into traffic patterns and usage statistics. NetFlow operates by monitoring IP traffic as it enters or exits a router or switch, and the data collected is used for various applications such as network monitoring, anomaly detection, and billing.

The protocol has evolved through several versions, with NetFlow v5 being one of the most widely used. NetFlow v5 defines a flow as a unidirectional sequence of packets between a given source and destination, sharing the same properties such as IP addresses, port numbers, protocol, and type of service (ToS) byte. The exporter sends flow records to a NetFlow collector, where the data is further analyzed or stored.

Key characteristics of NetFlow v5 include:

- **Fixed Record Format:** NetFlow v5 uses a fixed-length record format, which simplifies the process of flow export and reduces overhead.
- **Efficient Collection:** The flow export process is optimized for large-scale network traffic, which allows NetFlow to be widely used in both enterprise and service provider environments.
- **Real-Time Traffic Monitoring:** NetFlow provides a near real-time view of the traffic on a network, which is valuable for detecting traffic anomalies or attacks.

This project uses the NetFlow v5 protocol to export flow records from PCAP files, focusing on TCP flows. By adhering to the v5 specification, the exporter ensures compatibility with most NetFlow collectors and analyzers.

2.2 PCAP File Structure

The Packet Capture (PCAP) file format is the de facto standard for storing network packet data. PCAP files contain a log of network packets captured by a network analyzer, such as

tcpdump or Wireshark, and are used extensively in network analysis and security diagnostics.

A PCAP file is structured as follows:

- **Global Header:** Contains metadata about the capture, such as the file format version, network link type, and timestamp precision.
- **Packet Records:** Each record contains a timestamp and the length of the captured packet. This is followed by the actual packet data, which can include headers and payloads from layers 2 to 4 of the OSI model (e.g., Ethernet, IP, TCP).

The versatility of the PCAP format makes it ideal for a wide range of applications. PCAP files can be read by numerous software tools and provide a comprehensive view of network activity. For this project, PCAP files are used as the source of raw packet data, which is then parsed to identify TCP flows for NetFlow export.

2.2.1 PCAP Parsing in C/C++

Libraries like `libpcap` make it easier to read and process PCAP files in C/C++. `libpcap` allows for filtering specific types of traffic (e.g., only TCP packets) and provides functions to iterate through packet records in a PCAP file. For the purposes of this project, `libpcap` is used to extract packet headers and aggregate flow information based on the packet's source and destination addresses, port numbers, and timestamps.

2.3 Packet Capture and Traffic Analysis Tools

Various packet capture and traffic analysis tools are commonly used in network diagnostics, performance tuning, and security monitoring. Some of these tools influenced the development of this PCAP NetFlow exporter.

2.3.1 Wireshark

Wireshark is an open-source network protocol analyzer that allows users to capture and interactively browse the traffic running on a computer network. It supports many network protocols and has extensive filtering capabilities to isolate specific traffic flows.

For this project, Wireshark served as an inspiration for how traffic flows are visually represented and filtered. Its ability to analyze PCAP files helped define the structure of how PCAP files should be parsed and processed. Although Wireshark is highly feature-rich, it primarily focuses on real-time analysis, while this project emphasizes exporting historical flows from previously captured traffic in the NetFlow v5 format.

2.3.2 tcpdump

tcpdump is a lightweight, command-line tool used for capturing and analyzing network traffic. It allows users to capture packets that match specific criteria (e.g., port number, IP address, protocol) and store the captured data in PCAP files for further analysis. tcpdump is widely used for troubleshooting network issues and examining detailed network traffic.

This project leverages tcpdump's ability to generate PCAP files, which are then processed by the NetFlow exporter. tcpdump's filtering capabilities, using expressions based

on the Berkeley Packet Filter (BPF) syntax, also served as a model for designing the filtering capabilities of the PCAP NetFlow exporter, allowing specific traffic types to be isolated for export.

2.3.3 nfdump and nfsen

nfdump and nfsen are tools used for collecting, storing, and visualizing NetFlow data. nfdump provides command-line utilities for processing NetFlow records, while nfsen offers a web-based graphical interface for viewing traffic statistics and trends.

These tools were studied as part of the workflow for analyzing NetFlow data. While this project focuses on exporting NetFlow v5 records from PCAP files, the exported data is designed to be compatible with tools like nfdump and nfsen, allowing users to seamlessly integrate the exported flows into their existing NetFlow analysis pipelines.

2.4 Summary

In summary, the design and implementation of the PCAP NetFlow v5 exporter were heavily influenced by Cisco's NetFlow v5 protocol, the structure of PCAP files, and common packet capture and traffic analysis tools such as Wireshark, tcpdump, and nfdump. These tools and protocols provided the foundation for creating an efficient, configurable tool that can convert historical PCAP traffic into NetFlow records for further analysis and monitoring.

Chapter 3

Application Design

This chapter presents the design of the PCAP NetFlow v5 exporter, focusing on system architecture, data flow, and the main components of the application.

3.1 System Architecture

The architecture of the PCAP NetFlow v5 exporter is designed to efficiently process large PCAP files, aggregate flow information, and export it in the NetFlow v5 format to a designated collector. The system operates in three main stages: 1. **PCAP File Parsing**, 2. **Flow Aggregation**, and 3. **NetFlow Exporting**.

3.1.1 PCAP File Parsing

The first stage involves reading network traffic data from a PCAP file. This is achieved using the `libpcap` library, which allows the program to open a PCAP file and iterate over the captured packets. Each packet contains headers (e.g., Ethernet, IP, TCP) and, potentially, payload data. The program extracts essential header information such as:

- Source and destination IP addresses,
- Source and destination port numbers,
- Protocol type,
- Timestamps,
- Packet length.

Packets are filtered based on protocol (e.g., only TCP packets) before being passed to the next stage.

3.1.2 Flow Aggregation

After packet data is extracted, the program aggregates packets into flows. A flow is defined as a unidirectional sequence of packets that share the same:

- Source IP address,
- Destination IP address,

- Source and destination port numbers,
- Protocol,
- Type of service (ToS).

The exporter tracks active flows and updates flow records as new packets belonging to the same flow are processed. If no packets are seen for a flow after a predefined **inactive timeout** period, the flow is considered complete and is ready for export. The program also handles **active timeouts**, exporting long-running flows periodically.

3.1.3 NetFlow Exporting

Once a flow is completed (or has timed out), it is formatted into a NetFlow v5 record. The flow record includes:

- Source and destination IP addresses,
- Source and destination port numbers,
- Number of packets and bytes transferred,
- Start and end timestamps of the flow,
- Protocol and ToS information.

The exporter sends the formatted flow records to a NetFlow collector using UDP. The IP address and port of the collector are configurable by the user. The lightweight nature of the NetFlow v5 format and the use of UDP for transport ensure that the system can handle high volumes of traffic efficiently.

3.2 Data Flow

The data flow through the system is straightforward and follows these steps:

1.PCAP File Input: The user provides a PCAP file as input. The file contains recorded network packets from previous traffic captures. **2.Packet Parsing:** Using the `libpcap` library, the system reads each packet from the PCAP file, extracts relevant header information, and filters out non-TCP packets. **3.Flow Identification:** The system compares packet information to track and aggregate packets into flows based on their source and destination addresses, port numbers, and protocol. **4. Flow Aggregation:** Flows are updated as new packets are added, tracking the number of packets, total bytes, and timestamps. Once a flow is complete, it is marked for export. **5. Flow Export:** Completed flows are formatted into NetFlow v5 records and sent to the specified NetFlow collector via UDP.

The figure above illustrates the key steps in the data flow from the input of a PCAP file to the export of NetFlow records.

3.3 Component Breakdown

The application consists of several modular components, each responsible for a specific part of the process:

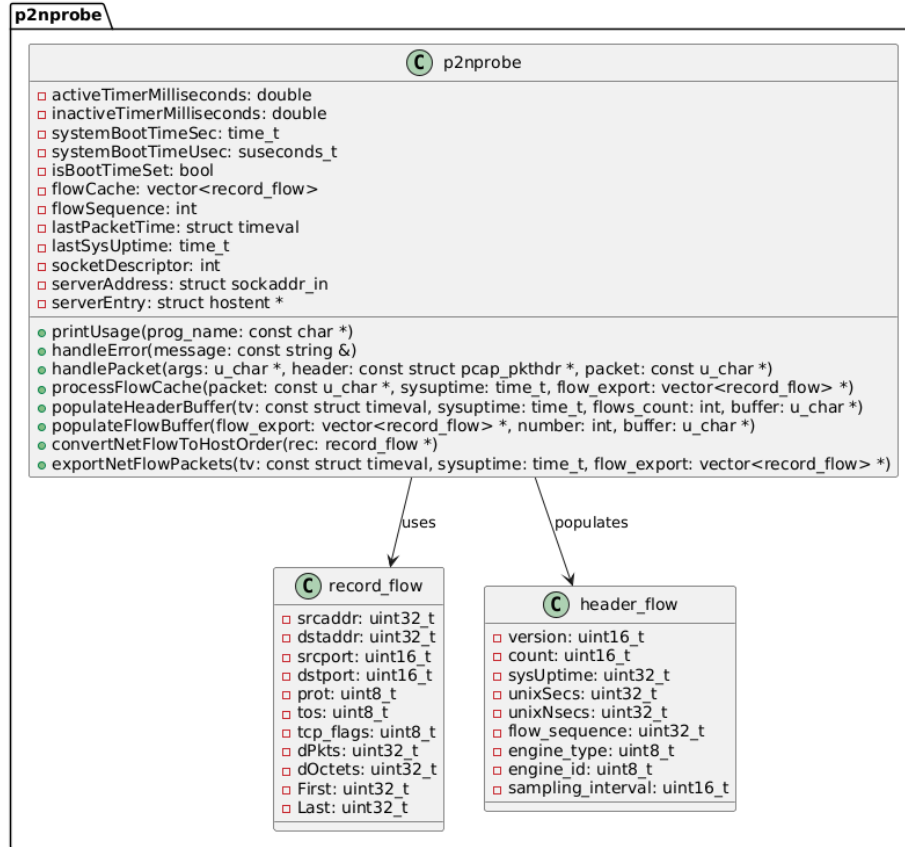


Figure 3.1: Data Flow Diagram for PCAP NetFlow Exporter

3.3.1 PCAP File Reader

The PCAP File Reader is responsible for opening and reading packets from the PCAP file using the `libpcap` library. It provides an abstraction for packet extraction, offering parsed header information to other components.

3.3.2 Flow Aggregator

The Flow Aggregator is responsible for tracking active flows and updating flow records as new packets are processed. It handles both active and inactive timeouts:

- **Active Timeout:** Ensures that long-running flows are periodically exported.
- **Inactive Timeout:** Exports flows that have not seen any new packets for a given period.

The Flow Aggregator stores flow data such as source/destination addresses, port numbers, protocol type, timestamps, and byte/packet counts.

3.3.3 NetFlow Exporter

The NetFlow Exporter formats aggregated flow records into the NetFlow v5 structure and sends them to a remote collector via UDP. The collector's IP address and port are

specified by the user via command-line parameters. The exporter is designed to handle high-throughput scenarios, ensuring that flow records are transmitted efficiently with minimal overhead.

3.3.4 Configuration Manager

The Configuration Manager handles user-defined settings such as:

- PCAP file location,
- Collector IP and port,
- Active and inactive timeout values.

These settings can be passed as command-line arguments when starting the program, allowing the user to customize the behavior of the exporter.

3.4 Summary

The application is designed with modularity in mind, dividing tasks such as packet parsing, flow aggregation, and NetFlow exporting into distinct components. The system processes PCAP files, aggregates TCP flow data, and exports the flows in the NetFlow v5 format, following an efficient, configurable workflow that can scale to handle large traffic captures.

Chapter 4

Implementation

4.1 Language and Libraries

This project is implemented in C/C++ on Unix systems. Key libraries used include:

- `libpcap` for reading PCAP files, enabling the capture and analysis of network traffic.
- `libnet` for creating and sending NetFlow v5 packets, which facilitates the transmission of flow data to collectors for further analysis.

4.2 Interesting Implementation Details

The implementation contains several noteworthy aspects that contribute to its functionality and performance:

4.2.1 Flow Aggregation Algorithm

The flow aggregation algorithm is designed to efficiently manage network flows as they are captured. It utilizes a cache to store active flows, which are identified by their source and destination IP addresses, ports, and protocol. The algorithm follows these key steps:

- **Flow Identification:** Each packet is analyzed to determine if it belongs to an existing flow. If a matching flow is found, its statistics (like packet count and byte count) are updated.
- **New Flow Creation:** If no matching flow exists, a new flow record is created and added to the cache.
- **Flow Timeout Management:** Flows are monitored for activity; if a flow remains inactive beyond a specified timeout, it is exported to the collector and removed from the cache.

4.2.2 Timeout Handling for Active and Inactive Flows

The implementation differentiates between active and inactive flows through timeout mechanisms:

- **Active Timeout:** This is the duration a flow is considered active after the last packet was received. If a flow remains inactive for this period, it is marked for export.

- **Inactive Timeout:** This timeout applies when a flow is still in the cache but hasn't had any packets sent for a longer period. Flows that exceed this timeout are also exported, ensuring that stale data does not persist in the cache.

These mechanisms allow the program to effectively manage memory and bandwidth, preventing overuse of resources while ensuring timely data transmission to collectors.

4.3 Program Structure

The program's source code is organized into several primary modules, each with distinct responsibilities:

- **Main Module (`p2nprobe.cpp`):** This is the entry point of the application, responsible for argument parsing, initializing components, and managing the flow of execution.
- **Flow Management Module:** This module handles flow aggregation, updating flow records, and timeout management. It contains functions like `handlePacket()`, which processes incoming packets and updates the flow cache.
- **Packet Export Module:** Responsible for preparing and sending NetFlow packets to the collector. Key functions include `exportNetFlowPackets()` and `populateHeaderBuffer()`, which ensure the correct formatting of data before transmission.
- **Utility Module:** Contains helper functions such as `printUsage()` for displaying command-line usage instructions and `handleError()` for managing error reporting.
- **Configuration Module:** This module manages configuration settings, including timeouts for active and inactive flows, ensuring that these parameters are set correctly based on user input.

The clear separation of responsibilities among these modules promotes maintainability and scalability, making it easier to extend the functionality in future iterations of the project.

Chapter 5

Program Usage

5.1 Installation

To install the `p2nprobe` program on Unix systems, ensure the following dependencies are installed:

5.1.1 Dependencies

- **GCC** or **Clang**: A C/C++ compiler.
- **libpcap**: A library for network packet capture. Install it via your package manager:

– On Ubuntu/Debian:

```
sudo apt-get install libpcap-dev
```

– On Fedora:

```
sudo dnf install libpcap-devel
```

5.1.2 Installation Steps

1. **Build the Program:**

```
make
```

2. **Next Step** (optional):

```
Be cool!
```

5.2 Command Line Interface (CLI)

5.2.1 Running the Program

```
./p2nprobe <host>:<port> <pcap_file_path> [-a <active_timeout> -i <inactive_timeout>]
```

5.2.2 Parameter Description

- `<pcap_file_path>`: Path to the PCAP file to be processed.
- `<host>`: IP address or domain name of the collector.
- `<port>`: Port of the collector to which the messages will be sent.
- `-a <active_timeout>`: Number of seconds to set the active timeout for flow export (default value if not specified is 60 seconds).
- `-i <inactive_timeout>`: Number of seconds to set the inactive timeout for flow export (default value if not specified is 60 seconds).

5.2.3 Example Command

An example command to run the program with parameters:

```
./p2nprobe 192.168.1.100:2055 /path/to/capture.pcap -a 120 -i 30
```

5.3 Usage Examples

5.3.1 Example 1: Basic Processing

To run p2nprobe with default settings:

```
./p2nprobe 192.168.1.100:2055 /path/to/capture.pcap
```

Expected Output: The program processes the PCAP file and sends flow information to the specified collector.

5.3.2 Example 2: Setting Active Timeout

To set the active timeout:

```
./p2nprobe 192.168.1.100:2055 /path/to/capture.pcap -a 120
```

Expected Output: Flow data is sent to the collector with the active timeout set to 120 seconds.

5.3.3 Example 3: Setting Inactive Timeout

To process a PCAP file with the inactive timeout setting:

```
./p2nprobe 192.168.1.100:2055 /path/to/capture.pcap -i 30
```

Expected Output: Flow data is sent to the collector with the inactive timeout set to 30 seconds.

5.3.4 Example 4: Using with DNS

If you want to process a PCAP file that contains DNS queries and send them to the collector, you can use the following command:

```
./p2nprobe [localhost:9995 or dnsserver:5555] /path/to/dns_capture.pcap -a 90 -i 45
```

Expected Output: The program processes DNS queries from the file `dns_capture.pcap` and sends flow information to the collector at `localhost:9995` or `dnsserver:5555` with an active timeout of 90 seconds and an inactive timeout of 45 seconds.

5.3.5 Example 5: Command Help

To display help about available commands and parameters:

```
./p2nprobe --help
```

Expected Output: A list of available commands and their descriptions will be displayed.
*It is not in our Task and it is not implemented, but it will show how to use app like an error.

Chapter 6

Testing and Results

6.1 Test Setup

The testing was conducted in an environment where multiple PCAP files were analyzed to evaluate the performance of the PCAP NetFlow v5 exporter. For this, Wireshark was utilized to capture and analyze the packet flows. The testing process involved sending several PCAP file flows to a NetFlow collector using a Python script.

6.2 Test Cases

The following test cases were designed to evaluate the functionality and accuracy of the exporter:

- **Correct handling of TCP flows:** Ensuring that the exporter accurately processes and identifies TCP flows from the PCAP files.
- **Proper aggregation of multiple packets into a single flow:** Verifying that multiple packets belonging to the same flow are aggregated correctly, resulting in a single flow record.
- **Accurate export to NetFlow v5 format:** Checking that the exported flows conform to the NetFlow v5 specifications and can be correctly interpreted by the collector.

6.3 Results

The results of the tests are as follows:

- **Performance Data:** The time taken to process each PCAP file was measured, providing insights into the efficiency of the exporter.
- **Accuracy of Exported Flows:** The exported flows were evaluated for accuracy, confirming that the TCP flows were correctly represented in the NetFlow v5 format.
- **Issues Encountered:** Any discrepancies or challenges faced during testing were documented for further analysis.

The testing commands used to compare the results of the collector with those generated by the exporter were:

```
% To send test data to the collector
sudo softflowd -r wslpcap.pcap -n localhost:9995 -v 5 -d -p tcp

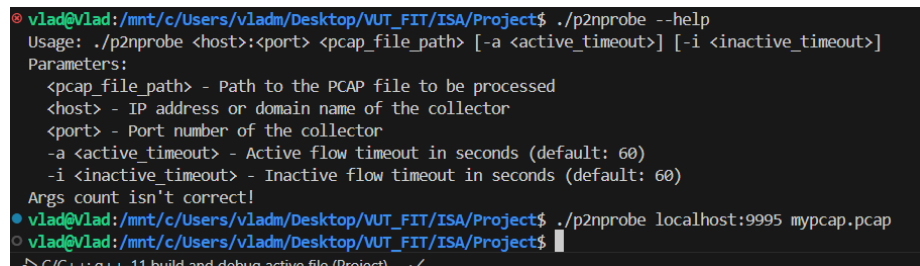
% To run the collector
nfcapd -l nfcapd/logs -p 9995

% To display the results
nfdump -r <File>
```

6.4 Visual Results

Figures in this section illustrate the testing process:

- Figure 6.1 shows how flows are sent from the console.
- Figure 6.2 displays how ‘nfcapd’ receives the flows.
- Figure 6.3 illustrates the packet capture in Wireshark, showing the processed flows.
- Figure 6.4 presents the results from the ‘nfdump’ command, comparing the output from the collector with that generated by the exporter.

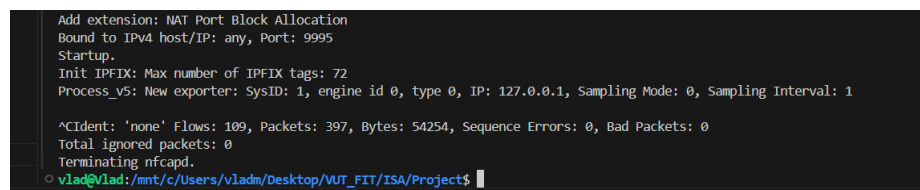


```

vlad@vlad:/mnt/c/Users/vladm/Desktop/VUT_FIT/ISA/Project$ ./p2nprobe --help
Usage: ./p2nprobe <host>:<port> <pcap_file_path> [-a <active_timeout>] [-i <inactive_timeout>]
Parameters:
  <pcap_file_path> - Path to the PCAP file to be processed
  <host> - IP address or domain name of the collector
  <port> - Port number of the collector
  -a <active_timeout> - Active flow timeout in seconds (default: 60)
  -i <inactive_timeout> - Inactive flow timeout in seconds (default: 60)
Args count isn't correct!
vlad@vlad:/mnt/c/Users/vladm/Desktop/VUT_FIT/ISA/Project$ ./p2nprobe localhost:9995 mypcap.pcap
vlad@vlad:/mnt/c/Users/vladm/Desktop/VUT_FIT/ISA/Project$

```

Figure 6.1: Sending Flows in Console



```

Add extension: NAT Port Block Allocation
Bound to IPv4 host/IP: any, Port: 9995
Startup.
Init IPFIX: Max number of IPFIX tags: 72
Process_v5: New exporter: SysID: 1, engine id 0, type 0, IP: 127.0.0.1, Sampling Mode: 0, Sampling Interval: 1
^CIdent: 'none' Flows: 109, Packets: 397, Bytes: 54254, Sequence Errors: 0, Bad Packets: 0
Total ignored packets: 0
Terminating nfcapd.
vlad@vlad:/mnt/c/Users/vladm/Desktop/VUT_FIT/ISA/Project$

```

Figure 6.2: Output from nfcapd

udp.port == 9995						
No.	Time	Source	Destination	Protocol	Length	Info
301	22.663617395	127.0.0.1	127.0.0.1	UDP	114	54679 → 9995 Len=72
302	22.663646590	127.0.0.1	127.0.0.1	UDP	114	54679 → 9995 Len=72
303	22.663651039	127.0.0.1	127.0.0.1	UDP	114	54679 → 9995 Len=72
304	22.663654465	127.0.0.1	127.0.0.1	UDP	114	54679 → 9995 Len=72
305	22.663790534	127.0.0.1	127.0.0.1	UDP	114	54679 → 9995 Len=72
306	22.663818798	127.0.0.1	127.0.0.1	UDP	114	54679 → 9995 Len=72
307	22.663823086	127.0.0.1	127.0.0.1	UDP	114	54679 → 9995 Len=72
308	22.663825882	127.0.0.1	127.0.0.1	UDP	114	54679 → 9995 Len=72
309	22.663828777	127.0.0.1	127.0.0.1	UDP	114	54679 → 9995 Len=72
310	22.663852673	127.0.0.1	127.0.0.1	UDP	114	54679 → 9995 Len=72
311	22.663856650	127.0.0.1	127.0.0.1	UDP	114	54679 → 9995 Len=72
312	22.663859796	127.0.0.1	127.0.0.1	UDP	162	54679 → 9995 Len=120
313	22.663862732	127.0.0.1	127.0.0.1	UDP	114	54679 → 9995 Len=72
314	22.663866487	127.0.0.1	127.0.0.1	UDP	114	54679 → 9995 Len=72
315	22.663890535	127.0.0.1	127.0.0.1	UDP	114	54679 → 9995 Len=72
316	22.663893290	127.0.0.1	127.0.0.1	UDP	114	54679 → 9995 Len=72
317	22.663916755	127.0.0.1	127.0.0.1	UDP	258	54679 → 9995 Len=216
318	22.663920682	127.0.0.1	127.0.0.1	UDP	114	54679 → 9995 Len=72
319	22.663923558	127.0.0.1	127.0.0.1	UDP	114	54679 → 9995 Len=72
320	22.663928848	127.0.0.1	127.0.0.1	UDP	114	54679 → 9995 Len=72
321	22.663931713	127.0.0.1	127.0.0.1	UDP	114	54679 → 9995 Len=72
322	22.663934468	127.0.0.1	127.0.0.1	UDP	114	54679 → 9995 Len=72
323	22.663970247	127.0.0.1	127.0.0.1	UDP	162	54679 → 9995 Len=120
324	22.664151361	127.0.0.1	127.0.0.1	UDP	162	54679 → 9995 Len=120
325	22.664179655	127.0.0.1	127.0.0.1	UDP	162	54679 → 9995 Len=120
326	22.664184184	127.0.0.1	127.0.0.1	UDP	162	54679 → 9995 Len=120
327	22.664208440	127.0.0.1	127.0.0.1	UDP	114	54679 → 9995 Len=72
328	22.664212408	127.0.0.1	127.0.0.1	UDP	114	54679 → 9995 Len=72
329	22.664215604	127.0.0.1	127.0.0.1	UDP	162	54679 → 9995 Len=120
330	22.664239108	127.0.0.1	127.0.0.1	UDP	114	54679 → 9995 Len=72
331	22.664369647	127.0.0.1	127.0.0.1	UDP	114	54679 → 9995 Len=72

▶ Frame 301: 114 bytes on wire (912 bits), 114 bytes captured (912 bits) on interface lo, id 0
 ▶ Ethernet II, Src: 00:00:00:00:00:00 (00:00:00:00:00:00), Dst: 00:00:00:00:00:00 (00:00:00:00:00:00)
 ▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
 ▶ User Datagram Protocol, Src Port: 54679, Dst Port: 9995
 ▶ Data (72 bytes)

```

0000  00 00 00 00 00 00 00 00 00 00 00 08 00 45 00  .....E.
0010  00 64 f1 e1 40 00 40 11 4a a5 7f 00 00 01 7f 00  -d-@-J-
0020  00 01 d5 97 27 0b 00 50 fe 63 00 05 00 01 00 00  -P-c-
0030  3e 2f 66 f6 cc ec 1d c3 d3 c8 00 00 00 00 00 00  >/f-
0040  00 00 17 d4 6e a8 0a 05 06 ff 00 00 00 00 00 00  -n-
0050  00 00 00 00 00 02 00 00 00 68 00 00 3e 2f 00 00  -h- >/-
0060  3e 2f 01 bb de 59 00 19 06 00 00 00 00 00 00 00  >/...Y-
0070  00 00
  
```

Figure 6.3: Wireshark Packet Capture

2024-09-27 17:19:28.251	INVALID	Ignore TCP	10.5.6.255:56174	->	162.159.135.234:443	0.0.0.0:0	->	0.0.0.0:0	
414	0								
2024-09-27 17:20:05.308	INVALID	Ignore TCP	13.89.179.8:443	->	10.5.6.255:56935	0.0.0.0:0	->	0.0.0.0:0	7
227	0								
2024-09-27 17:20:06.811	INVALID	Ignore TCP	10.5.6.255:56935	->	13.89.179.8:443	0.0.0.0:0	->	0.0.0.0:0	
40	0								
2024-09-27 17:19:49.941	INVALID	Ignore TCP	10.5.6.255:56928	->	147.229.180.115:7680	0.0.0.0:0	->	0.0.0.0:0	
260	0								
2024-09-27 17:18:53.593	INVALID	Ignore TCP	149.154.167.92:443	->	10.5.6.255:56717	0.0.0.0:0	->	0.0.0.0:0	8
538	0								
2024-09-27 17:18:53.646	INVALID	Ignore TCP	10.5.6.255:56717	->	149.154.167.92:443	0.0.0.0:0	->	0.0.0.0:0	5
879	0								
2024-09-27 17:19:48.330	INVALID	Ignore TCP	10.5.6.255:56833	->	140.82.114.26:443	0.0.0.0:0	->	0.0.0.0:0	
111	0								
2024-09-27 17:19:48.429	INVALID	Ignore TCP	140.82.114.26:443	->	10.5.6.255:56833	0.0.0.0:0	->	0.0.0.0:0	
158	0								
2024-09-27 17:19:57.787	INVALID	Ignore TCP	10.5.6.255:56934	->	88.221.92.9:443	0.0.0.0:0	->	0.0.0.0:0	
895	0								
2024-09-27 17:19:57.797	INVALID	Ignore TCP	88.221.92.9:443	->	10.5.6.255:56934	0.0.0.0:0	->	0.0.0.0:0	7
699	0								
2024-09-27 17:19:58.587	INVALID	Ignore TCP	10.5.6.255:56172	->	35.186.224.45:443	0.0.0.0:0	->	0.0.0.0:0	
123	0								
2024-09-27 17:19:58.601	INVALID	Ignore TCP	35.186.224.45:443	->	10.5.6.255:56172	0.0.0.0:0	->	0.0.0.0:0	
120	0								
2024-09-27 17:19:59.006	INVALID	Ignore TCP	10.5.6.255:56178	->	35.186.224.45:443	0.0.0.0:0	->	0.0.0.0:0	
108	0								
2024-09-27 17:19:59.025	INVALID	Ignore TCP	35.186.224.45:443	->	10.5.6.255:56178	0.0.0.0:0	->	0.0.0.0:0	
104	0								
2024-09-27 17:19:59.610	INVALID	Ignore TCP	10.5.6.255:56179	->	34.158.1.133:4070	0.0.0.0:0	->	0.0.0.0:0	
91	0								
2024-09-27 17:19:59.638	INVALID	Ignore TCP	34.158.1.133:4070	->	10.5.6.255:56179	0.0.0.0:0	->	0.0.0.0:0	
51	0								
2024-09-27 17:20:00.359	INVALID	Ignore TCP	10.5.6.255:56181	->	142.250.102.188:5228	0.0.0.0:0	->	0.0.0.0:0	
41	0								
2024-09-27 17:20:00.380	INVALID	Ignore TCP	142.250.102.188:5228	->	10.5.6.255:56181	0.0.0.0:0	->	0.0.0.0:0	
52	0								

Summary: total flows: 109, total bytes: 54254, total packets: 397, avg bps: 5548, avg pps: 5, avg bpps: 136
 Time window: 2024-09-27 17:18:52 - 2024-09-27 17:20:10
 Total flows processed: 109, Blocks skipped: 0, Bytes read: 8848
 Sys: 0.004s flows/second: 22544.0 Wall: 0.007s flows/second: 15384.6
 vlad@Vlad:/mnt/c/Users/vlad/Desktop/VUT_FIT/ISA/Projects

Figure 6.4: Console Output from nfdump

Chapter 7

Conclusion

In summary, this project successfully implemented a PCAP NetFlow v5 exporter, demonstrating the ability to efficiently process and export TCP flows from PCAP files. The exporter was able to handle large datasets, accurately aggregate packets, and conform to the NetFlow v5 specifications, making it a reliable tool for network traffic analysis.

While the project met its primary objectives, there were some limitations encountered during testing. One notable limitation was the occasional discrepancy in flow timing when handling large PCAP files, which could lead to slight inaccuracies in the timestamps of the exported flow records. Another challenge was optimizing the performance for very high-throughput scenarios, where the processing time could be further reduced with additional refinements. Moreover, handling fragmented packets more effectively is an area that could be improved, as these cases were not fully addressed in the current implementation.

For future work, several areas of improvement and exploration could be considered:

- **Performance Optimization:** Implementing more efficient algorithms to handle large volumes of network traffic and reduce processing time, especially for high-throughput environments.
- **Support for More Protocols:** Expanding the tool to support other network protocols beyond TCP, such as UDP and ICMP, to make the exporter more versatile.
- **Improved Fragmentation Handling:** Enhancing the handling of IP fragment re-assembly to ensure that all types of fragmented packets are correctly processed and exported.
- **Real-Time Flow Exporting:** Incorporating real-time flow exporting capabilities to continuously process live network traffic instead of relying solely on PCAP file inputs.
- **Extended Testing:** Conducting more extensive tests with diverse network traffic types and various real-world scenarios to evaluate the robustness and reliability of the tool.

By addressing these areas, the tool could be further developed into a comprehensive network analysis utility, capable of providing deeper insights into network behavior across various environments.

Chapter 8

Literature

- Cisco Systems, Inc. „Introduction to Cisco IOS NetFlow - A Technical Overview.“ 2012.
- McCanne, Steven, and Van Jacobson. „The BSD Packet Filter: A New Architecture for User-level Packet Capture.“ 1993.
- [IBM Documentation: NetFlow v5 Formats](#)
- [Cisco Documentation: NetFlow Format](#)
- [Wikipedia: NetFlow](#)