BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

CUSTOM ATTRIBUTE EXPORT EXTENSION FOR NETFLOW PROBE

ROZŠÍŘENÍ EXPORTU VLASTNÍCH ATRIBUTŮ PRO SONDU NETFLOW

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **Bc. BRANISLAV MATEÁŠ**
AUTOR PRÁCE

SUPERVISOR **Ing. MATĚJ GRÉGR, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2023

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Master's Thesis Assignment

156847

Institut:	Department of Information Systems (DIFS)
Student:	**Mateáš Branislav, Bc.**
Programme:	Information Technology and Artificial Intelligence
Specialization:	Computer Networks
Title:	**Custom attribute export extension for NetFlow probe**
Category:	Networking
Academic year:	2023/24

Assignment:

1. Learn about network flow monitoring and available NetFlow exporters on the market.
2. Study the creation of kernel modules for the GNU/Linux kernel, focusing on the ipt-netflow module for exporting NetFlow.
3. Design and implement an extension to the ipt-netflow kernel module to export custom attributes. Implement export of a custom attribute calculating packet size variance in the flow.
4. Measure the impact of the implementation on CPU load.

Literature:

- Claise, B. (2004). *Cisco systems netflow services export version 9* (No. rfc3954).
- Rosen, R. Linux Kernel Networking: Implementation and Theory. New York, NY: Apress, 2014. ISBN 9781430261964.
- Benvenuti, Ch. Understanding Linux Network Internals. O'Reilly Media, Inc., 2005. ISBN 9780596002558
- Bharadwaj, R. Mastering Linux Kernel development, Birmingham, UK, 2017. ISBN 9781785886133
- Data export module for Linux kernel, https://github.com/aabc/ipt-netflow

Requirements for the semestral defence:
1., 2.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Grégr Matěj, Ing., Ph.D.**
Head of Department:	Kolář Dušan, doc. Dr. Ing.
Beginning of work:	1.11.2023
Submission deadline:	17.5.2024
Approval date:	30.10.2023

Abstract

This master's thesis delves into the augmentation of NetFlow kernel module by implementing a custom attribute export feature focused on capturing variance. The output of this thesis is a tutorial on how to extend the implementation of the ipt_netflow module to export a custom netflow attribute, and the implementation itself extended to export the Variance netflow attribute. Variance adds a new dimension to network flow data that is particularly valuable for anomaly detection and network performance monitoring.

Abstrakt

Táto diplomová práca sa zaoberá rozšírením modulu NetFlow implementáciou vlastnej funkcie exportu atribútov zameranej na zachytávanie odchýlok. Výstupom tejto práce je návod na rozšírenie implementácie modulu ipt_netflow o export vlastného atribútu netflow a samotná implementácia rozšírená o export atribútu odchýlky velkosti paketov. Odchýlka veľkosti paketov pridáva údajom o sieťových tokoch nový rozmer, ktorý je obzvlášť cenný pre detekciu anomálií a monitorovanie výkonnosti siete.

Keywords

netflow, linux, kernel, variance

Klíčová slova

netflow, linux, kernel, variance

Reference

MATEÁŠ, Branislav. *Custom attribute export extension for NetFlow probe.* Brno, 2023. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Matěj Grégr, Ph.D.

Rozšířený abstrakt

Táto diplomová práca sa zaoberá návrhom a rozšírením implementácie modulu ipt_netflow pre kernel linuxu. Cieľom tejto diplomovej práce je ponoriť sa do implementácie modulu ipt_netflow, analyzovať a pochopiť ako rozšíriť túto implementáciu o export vlastných atribútov netflow protokolu. Súčasť diplomovej práce je teda poskytnúť čitateľovi návod ako do tejto implementácie pridať vlastný atribút, kde je presne popísané aké súbory, funkcie a dátové štruktúry musia byť upravené a doplnené pre dosiahnutie exportu vlastného netflow atribútu. Pre demonštrovanie, ako takýto netflow atribút pridať, je súčasťou aj samotné rozšírenie ipt_netflow modulu o export atribútu odchýlky veľkosti poaketov, ktorý má široké využitie v detekcii sieťových útokov.

Custom attribute export extension for NetFlow probe

Declaration

I hereby declare that this Master's thesis was prepared as an original work by the author under the supervision of Ing. Matěj Grégr Ph.D.. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Branislav Mateáš
May 14, 2024

Contents

List of Figures

Chapter 1

Introduction

This master's thesis delves into the augmentation of NetFlow kernel module by implementing a custom attribute export feature focused on capturing variance. NetFlow, a widely utilized protocol for network traffic analysis, traditionally provides insights into communication patterns. However, the incorporation of a variance-based custom attribute export adds a new dimension to network flow data, particularly valuable for anomaly detection and network performance monitoring.

The primary objective of this research is to design, implement, and evaluate a custom attribute export mechanism within the NetFlow kernel module that specifically captures variance in network flow attributes. Variance, as a statistical measure, can reveal deviations from normal patterns, making it a powerful tool for identifying potential anomalies or security threats within a network.

The methodology involves an in-depth exploration of existing NetFlow implementations, kernel module development techniques, and statistical analysis methods. The design phase focuses on specifying the attributes for which variance will be calculated and developing the necessary algorithms within the NetFlow module to export this information.

The implementation process includes coding the variance calculation logic and integrating it seamlessly into the NetFlow kernel module. Rigorous testing and validation will be conducted to ensure the accuracy and effectiveness of the variance-based custom attribute export. Performance metrics, such as computational overhead and resource utilization, will be evaluated to gauge the impact on overall system performance.

The thesis also explores real-world use cases to demonstrate the practical applications of the enhanced NetFlow module with variance-based attribute export. This involves deploying the solution in diverse network environments and assessing its ability to identify anomalies, fluctuations, or unexpected patterns that may indicate security incidents or performance issues.

In conclusion, this research contributes to the advancement of network flow analysis by introducing a variance-based custom attribute export feature to the NetFlow kernel module. The outcomes of this study aim to enhance the capabilities of network administrators and security analysts in detecting and responding to network anomalies, ultimately bolstering the resilience and security posture of the monitored network infrastructure.

Chapter 2

Netflow

Network monitoring approaches have been proposed and developed throughout the years, each of them serving a different purpose. They can generally be classified into two categories: active and passive. Active approaches, such as implemented by tools like Ping and Traceroute, inject traffic into a network to perform different types of measurements. Passive approaches observe existing traffic as it passes by a measurement point and therefore observe traffic generated by users. One passive monitoring approach is packet capture. This method generally provides most insight into the network traffic, as complete packets can be captured and further analyzed. However, in high-speed networks packet capture requires expensive hardware and substantial infrastructure for storage and analysis. Another passive network monitoring approach that is more scalable for use in high-speed networks is flow export, in which packets are aggregated into flows and exported for storage and analysis. A flow is defined in [7] as "a set of IP packets passing an observation point in the network during a certain time interval, such that all packets belonging to a particular flow have a set of common properties". These common properties may include packet header fields, such as source and destination IP addresses and port numbers, packet contents, and meta-information.

Cisco Systems' NetFlow services give network administrators access to IP flow information from their data networks. Network elements (routers and switches) collect flow data and send it to collectors. The collected data enables fine-grained metering, resulting in highly flexible and detailed resource usage accounting.

2.1 Capturing packet flow

Network data representation forms, as seen in a small network linked to the Internet via a forwarding device in figure 2.1. The upper host sends packets 4 and 3, and receives packet 6, whereas the lower host sends packet 2. Packet 1 is sent, and packet 5 is received by the central forwarding device. The network's observation point allows for the following observations: (a) packet capture: the contents (control information and, optionally, payload) of each network packet are recorded; (b) device statistics: only statistics of packets passing the observation point are recorded; and (c) flows: packets belonging to a flow (common property is the packet's outline style) are aggregated (number of packets in this example).

The architecture of typical flow monitoring setups consists of several stages, each of which is shown in figure 2.2.

Packet observation process has 5 stages [10]:

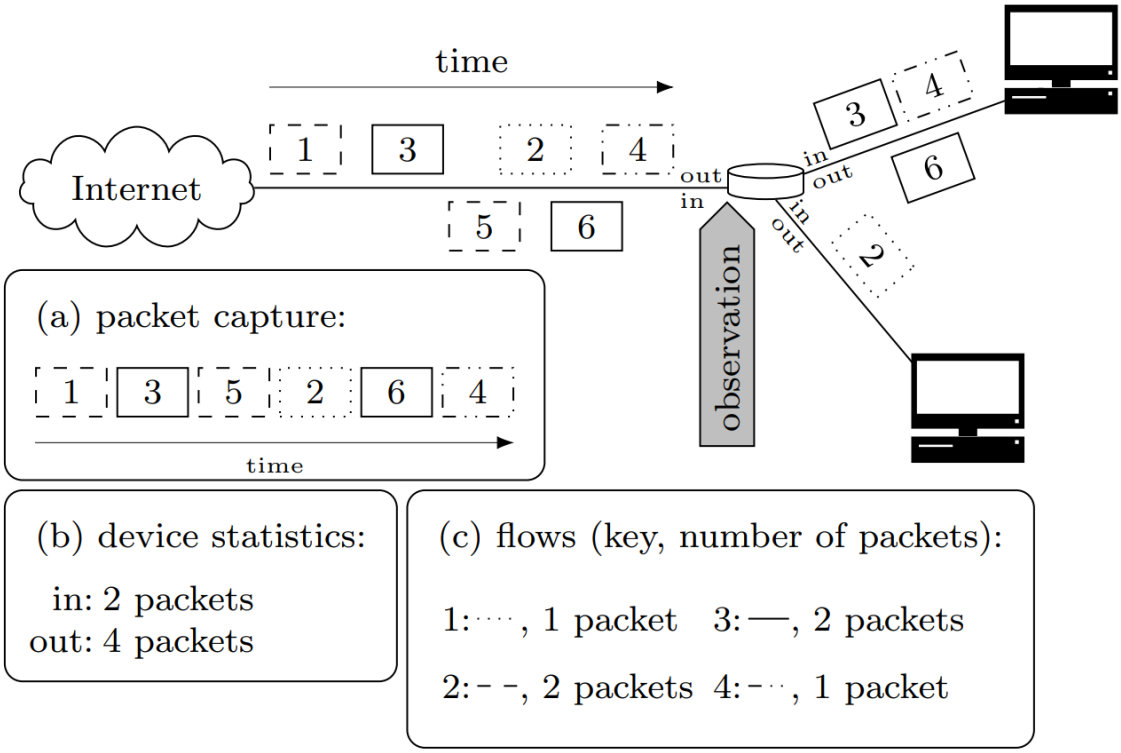

Figure 2.1: Flow capture observation [14]

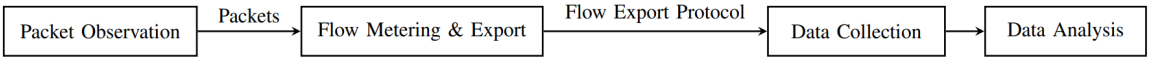

Figure 2.2: Flow monitoring setup [10]

- **Packet capture** - before any packet pre-processing can be performed, packets must be read from the line. This step, packet capture, is the first in the architecture and typically carried out by a Network Interface Card (NIC). Before packets are stored in on-card reception buffers and later moved to the receiving host's memory, they have to pass several checks when they enter the card, such as checksum error checks.
- **Timestamping** - accurate packet timestamps are essential for many processing functions and analysis applications. For example, when packets from different observation points have to be merged into a single dataset, they will be ordered based on their timestamps. Timestamping performed in hardware upon packet arrival avoids delays as a consequence of forwarding latencies to software, resulting in an accuracy of up to 100 nanoseconds in the case of the IEEE 1588 protocol, or even better. Unfortunately, hardwarebased timestamping is typically only available on special NICs using Field Programmable Gate Arrays (FPGAs), and most commodity cards perform timestamping in software. However, software-based clock synchronization by means of the Network Time Protocol (NTP) or the Simple Network Time Protocol (SNTP) usually provides an accuracy in the order of 100 microseconds. For more information about timestamping you can refer to [11] [12].

- **Truncation** - selects only those bytes of a packet that fit into a preconfigured snapshot length. This reduces the amount of data received and processed by a capture application, and therefore also the number of computation cycles, bus bandwidth and memory used to process the network traffic. For example, flow exporters traditionally only rely on packet header fields and ignore packet payloads.
- **Packet Sampling** S_i - „aims at reducing the load of subsequent stages or processes and, consequently, to reduce the consumption of bandwidth, memory and computation cycles“[9]. Therefore, sampling should be used whenever it is expected that the number of monitored packets will overload the following stage. The ultimate goal is to turn the uncontrolled loss of packets caused by overload into a controlled one by using sampling.
- **Packet Filtering** F_i - deterministically „separate all the packets having a certain property from those not having it“ [15]. Similar to sampling, filtering can be used to reduce the amount of data to be processed by the subsequent stages.

The Flow Metering & Export stage is where packets are aggregated into flows and flow records are exported, which makes it key to any flow monitoring system. The packet aggregation is performed based on Information Elements that define the layout of a flow. After aggregation, an entry per flow is stored in a flow cache until a flow is considered to have terminated and the entry is expired. After one or more optional flow-based sampling and filtering functions, flow records have to be encapsulated in messages. This is where **Netflow** comes in.

Data Collection - Flow collectors are an integral part of flow monitoring setups, as they receive, store, and pre-process flow data from one or more flow exporters in the network. Data collection is performed by one or more Collecting Processes within flow collectors. Common pre-processing tasks are data compression, aggregation, data anonymization, filtering, and summary generation. The functionality and performance provided by flow collectors depend strongly on the underlying data storage format, as this defines how and at which speed data can be read and written. Although flow data often has to be stored for a long time (e.g., to comply with data retention laws), it can be useful to keep data in-memory. This is mostly the case when flow data has to be analyzed on-the-fly, and only results have to be stored. In those situations, one can benefit from the high performance of volatile storage. An example use case is the generation of a time-series in which only the time-series data itself has to be stored. When data has to be stored beyond the time needed for processing, it has to be moved to persistent storage. This, however, results in a bottleneck, due to the difference in speed between volatile and persistent storage. Depending on the system facilitating the flow collection, one may consider to compress data before moving it to persistent storage.

Data Analysis is the final stage in a flow monitoring setup, where the results of all previous stages come together. We distinguish between three application areas for data analysis, which are widely used for classifying analysis software:

1. **Flow Analysis & Reporting** - given that flow export devices are commonly deployed at strategical locations in a network where traffic from a large number of hosts can be observed, the resulting data provides a comprehensive set of connections sum-

maries. Flow Analysis & Reporting is the most basic functionality provided by flow analysis applications and typically provides the following functionality:

- Browsing and filtering flow data
- Statistics overview - **Variance**
- Reporting and alerting

2. **Threat Detection** - when flow data is used for threat detection, we can distinguish between roughly two types of uses. First, flow data may be used purely for analyzing which host has communicated with which each other host (i.e., forensics), potentially including summaries of the number of packet and bytes involved, the number of connections, etc. The second utilizes the definition of a flow for analyzing certain types of threats, which allows for modeling threats in terms of network behavior. In the remainder of this section, we discuss an example of both types.

3. **Performance Monitoring** - performance monitoring aims at observing the status of services running on the network. Typical metrics that such data analysis applications report include Round-Trip-Time (RTT), delay, jitter, response time, packet loss and bandwidth usage. Performance monitoring applications post-process flow data and show a set of metrics per target service, to verify ServiceLevel Agreement (SLA) compliance and, ultimately, reveal network events and their impact on end-user experience. As for the other types of data analysis, the greatest strength of monitoring performance using flow measurements comes from the strategical vantage points from where flow measurements are usually taken. As a comparison, monitoring applications by means of client instrumentation requires the installation of agents in client devices, which makes the measurement environment not only less convenient to set up, but also harder to be managed. However, as we will exemplify next, flow measurements sometimes provide only a coarse approximation of common performance metrics, since such metrics are generally not directly measured and exported.

2.2 Netflow

NetFlow is a feature that was introduced on Cisco routers around 1996 that provides the ability to collect IP network traffic as it enters or exits an interface. It optimizes the network infrastructure, reducing operation costs and improving capacity planning and security incident detection with increased flexibility and scalability. The ability to characterize IP traffic and identify its source, traffic destination, timing, and application information is critical for network availability, performance, and troubleshooting. The monitoring of IP traffic flows increases the accuracy of capacity planning and ensures that resource allocation supports organizational goals. NetFlow helps users determine how to optimize resource usage, plan network capacity, and identify the optimal application layer for Quality of Service (QoS). It plays a vital role in network security by detecting Denial of Service (DoS) attacks and network-propagated worms.

2.3 Netflow architecture

NetFlow solutions generally have three main components[4]:

- **Netflow exporter** - A NetFlow-enabled device, usually a router or firewall, operates as a flow exporter and collects flow information. It aggregates data packets into flows and periodically exports NetFlow records via User Datagram Protocol (UDP) to one or more NetFlow collectors.

  The exporter identifies a flow as a unidirectional packet stream with at least one of these elements in common: input interface port, the source IP address and destination address, source port, destination port number, Layer-3 protocol field or type of service. A flow is ready for NetFlow export when it's inactive for a set period of time. It's also ready when a TCP flag, such as FIN or RST, shows that the flow has ended.

- **Netflow collector** - A Netflow collector can be hardware- or software-based, although software-based tools are more commonly used. NetFlow collectors receive the aggregated flow record data from flow exporter tools, and then preprocess and store it.

- **Netflow analyzer** - A NetFlow analyzer is a tool that processes and analyzes NetFlow records received and stored by a flow collector. It turns data into reports and alerts that provide insight on bandwidth usage, bandwidth hogs, traffic patterns, application usage and other performance metrics that may identify security threats and performance problems. This traffic flow analysis lets you create a picture of your network traffic and volume.

2.4 Netflow v5

NetFlow version 5 is an earlier iteration of the NetFlow protocol and represents a standardized approach to collecting and exporting information about IP traffic flows. In NetFlow v5, the focus is on fundamental flow information, encompassing details such as source and destination IP addresses, source and destination ports, as well as the quantity of bytes and packets associated with each flow. This version provides a basic but effective means of monitoring and analyzing network traffic. The packet format is fixed and comes with header (described in table 2.1) and record (described in table 2.2). Therefore it's always the same and hence is easy to decipher for most Netflow collection and network traffic reporting packages.

Bytes	**Content**	**Description**
0-1	version	Netflow export format version number
2-3	count	Number of flows exported in this packet (1-30)
4-7	sys_uptime	Current time in milliseconds since the export device booted
8-11	unix_secs	Current cound of second since 0000 UTC 1970
12-15	unix_nsecs	Residual nanoseconds since 0000 UTC 1970
16-19	flow_sequence	Sequence counter of total flows seen
20	engine_type	Type of flow-switching engine
21	engine_id	Slot number of the flow-switching engine
22-23	sampling_interval	First 2 bits hold the sampling mode; remaining 14 bits hold value of sampling interval

Table 2.1: Netflow v5 Flow header format [13]

Bytes	Content	Description
0-3	srcaddr	Source IP address
4-7	dstaddr	Destination IP address
8-11	nexthop	IP address of next-hop router
12-13	input	Simpe Network Manager Protocol (SNMP) index of input interface
14-15	output	SNMP index of output interface
16-19	dPkts	Packets in the flow
20-23	dOctets	Total number of Layer 3 bytes in the packets of the flow
24-27	first	SysUptime at start of the flow
28-31	last	SysUptime at the time the last packet of the flow was received
32-33	srcport	TCP/UDP source port number or equivalent
34-35	dstport	TCP/UDP destination port number or equivalent
36	pad1	Unused (0) bytes
37	tcp_flags	Cumulative OR of TCP flags
38	prot	IP protocol type (for example, TCP = 6; UDP = 17)
39	tos	IP tye of service (ToS)
40-41	src_as	Autonomous system number (ASN) of the source, either origin or peer
42-43	dsc_as	ASN of destination, either origin or peer
44	src_mask	Source address prefix mask bits
45	dst_mask	Destination address prefix mask bits
46-47	pad2	Unused (0) bytes

Table 2.2: Netflow v5 Flow record format [13]

2.5 Netflow v9

NetFlow version 9 marks an evolution in the NetFlow protocol, introducing enhancements and greater flexibility. Unlike its predecessor Netflow v5, NetFlow v9 utilizes a template-based approach for flow export. This innovation allows for a broader range of information elements to be included in the flow records. NetFlow v9's flexibility is particularly advantageous in scenarios where a diverse set of data attributes needs to be captured for comprehensive network analysis. While offering increased versatility, NetFlow v9 is not as standardized as v5, as it relies on templates to define the format of flow records. This adaptability, however, makes it a powerful choice for environments with varying flow record requirements.

2.5.1 Netflow v9 packet layout

The NetFlow Version 9 record format consists of a packet header followed by at least one or more template or data FlowSets. A template FlowSet provides a description of the fields that will be present in future data FlowSets. These data FlowSets may occur later within the same export packet or in subsequent export packets. Template and data FlowSets can be intermingled within a single export packet, as illustrated in Table 2.3. The possible combinations that can occur in an export packet:

- An export packet that consists of interleaved template and data FlowSets-A collector device should not assume that the template IDs defined in such a packet have any

Packet header	Template FlowSet	Data FlowSet	.....	Template FlowSet	Data FlowSet

Table 2.3: Netflow v9 export packet [3]

specific relationship to the data FlowSets within the same packet. The collector must always cache any received templates, and examine the template cache to determine the appropriate template ID to interpret a data record.

- An export packet consisting entirely of data FlowSets-after the appropriate template IDs have been defined and transmitted to the collector device, most of the export packets will consist solely of data FlowSets.
- An export packet consisting entirely of template FlowSets-although this case is the exception, it is possible to receive packets containing only template records. Ordinarily, templates are „piggybacked“ onto data FlowSets. However, in some instances only templates are sent. When a router first boots up or reboots, it attempts to synchronize with the collector device as quickly as possible. The router may send template FlowSets at an accelerated rate so that the collector device has sufficient information to interpret any subsequent data FlowSets. Also, template records have a limited lifetime, and they must be periodically refreshed. If the refresh interval for a template occurs and there is no appropriate data FlowSet that needs to be sent to the collector device, an export packet consisting solely of template FlowSets is sent.

2.5.2 Netflow v9 header layout

The format of the NetFlow Version 9 packet header remains relatively unchanged from previous versions. It is based on the NetFlow Version 5 packet header and is illustrated in Table 2.4. Table 2.5 gives field descriptions.

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31	
Version	Count
System uptime	
UNIX seconds	
Packet sequence	
Source ID	

Table 2.4: Netflow v9 header [6]

2.5.3 NetFlow Version 9 Template FlowSet layout

One of the key elements in the new NetFlow Version 9 format is the template FlowSet. Templates greatly enhance the flexibility of the NetFlow record format, because they allow a NetFlow collector or display application to process NetFlow data without necessarily knowing the format of the data in advance. Templates are used to describe the type and length of individual fields within a NetFlow data record that match a template ID. The format of the template FlowSet is described in Table 2.6, and the field descriptions are given in 2.7.

Field Name	Value
Version	The version of NetFlow records exported in this packet; for Version 9, this value is 0x0009
Count	Number of FlowSet records (both template and data) contained within this packet
System Uptime	Time in milliseconds since this device was first booted
UNIX Seconds	Seconds since 0000 Coordinated Universal Time (UTC) 1970
Packet Sequence	Incremental sequence counter of all export packets sent by this export device; this value is cumulative, and it can be used to identify whether any export packets have been missed
Source ID	The Source ID field is a 32-bit value that is used to guarantee uniqueness for all flows exported from a particular device. (The Source ID field is the equivalent of the engine type and engine ID fields found in the NetFlow Version 5 and Version 8 headers). The format of this field is vendor specific. In the Cisco implementation, the first two bytes are reserved for future expansion, and will always be zero. Byte 3 provides uniqueness with respect to the routing engine on the exporting device. Byte 4 provides uniqueness with respect to the particular line card or Versatile Interface Processor on the exporting device. Collector devices should use the combination of the source IP address plus the Source ID field to associate an incoming NetFlow export packet with a unique instance of NetFlow on a particular device.

Table 2.5: Netflow v9 header fields description [6]

2.5.4 NetFlow Version 9 Data FlowSet Format

The format of the data FlowSet is described in Table 2.8, and the field descriptions are given in Table 2.9.

2.5.5 Example

Full example of Netflow v9 export packet is shown in Figure 2.3

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FlowSet ID = 0															
Length															
Template ID															
Field Count															
Field 1 Type															
Field 1 Length															
Field 2 Type															
Field 2 Length															
. . .															
Field N Type															
Field N Length															
Template ID															
Field Count															
Field 1 Type															
Field 1 Length															
Field 2 Type															
Field 2 Length															
. . .															
Field N Type															
Field N Length															

Table 2.6: NetFlow Version 9 Template FlowSet Format [6]

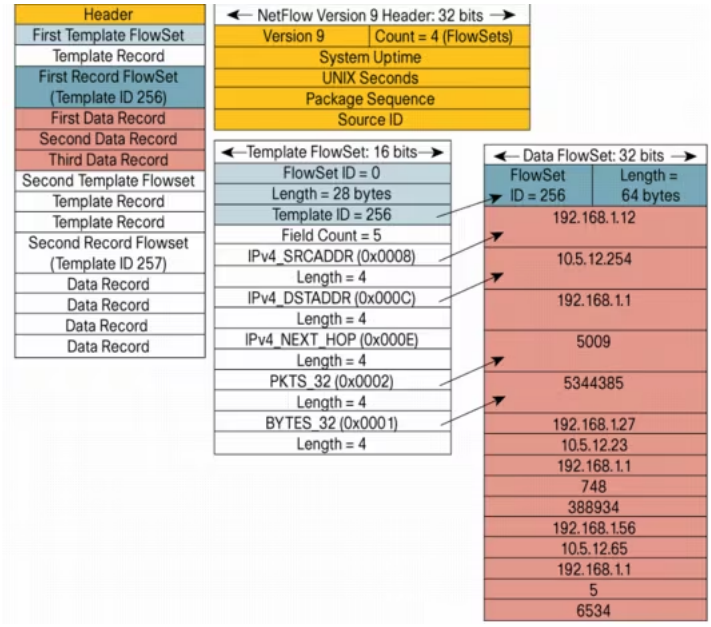

Figure 2.3: Netflow v9 export example [1]

Field Name	Value
FlowSet ID	The FlowSet ID is used to distinguish template records from data records. A template record always has a FlowSet ID in the range of 0-255. Currently, the template record that describes flow fields has a FlowSet ID of zero and the template record that describes option fields (described below) has a FlowSet ID of 1. A data record always has a nonzero FlowSet ID greater than 255.
Length	Length refers to the total length of this FlowSet. Because an individual template FlowSet may contain multiple template IDs (as illustrated above), the length value should be used to determine the position of the next FlowSet record, which could be either a template or a data FlowSet.Length is expressed in Type/Length/Value (TLV) format, meaning that the value includes the bytes used for the FlowSet ID and the length bytes themselves, as well as the combined lengths of all template records included in this FlowSet.
Template ID	As a router generates different template FlowSets to match the type of NetFlow data it will be exporting, each template is given a unique ID. This uniqueness is local to the router that generated the template ID. Templates that define data record formats begin numbering at 256 since 0-255 are reserved for FlowSet IDs.
Field Count	This field gives the number of fields in this template record. Because a template FlowSet may contain multiple template records, this field allows the parser to determine the end of the current template record and the start of the next.
Field Type	This numeric value represents the type of the field. The possible values of the field type are vendor specific. Cisco supplied values are consistent across all platforms that support NetFlow Version 9. At the time of the initial release of the NetFlow Version 9 code (and after any subsequent changes that could add new field-type definitions), Cisco provides a file that defines the known field types and their lengths. The currently defined field types are detailed in Table 6.
Field Length	This number gives the length of the above-defined field, in bytes.

Table 2.7: NetFlow Version 9 Template FlowSet Field Descriptions [6]

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FlowSet ID = Template ID															
Length															
Record 1 - Field 1 value															
Record 1 - Field 2 value															
Record 1 - Field 3 value															
Record 1 - Field 4 value															
. . .															
Record 1 - Field N value															
Record 2 - Field 1 value															
Record 2 - Field 2 value															
Record 2 - Field 3 value															
. . .															
Record 2 - Field N value															
. . .															
Padding															

Table 2.8: NetFlow Version 9 Data FlowSet Format [6]

Field Name	Value
FlowSet ID = Template ID	A FlowSet ID precedes each group of records within a NetFlow Version 9 data FlowSet. The FlowSet ID maps to a (previously received) template ID. The collector and display applications should use the FlowSet ID to map the appropriate type and length to any field values that follow.
Length	This field gives the length of the data FlowSet. Length is expressed in TLV format, meaning that the value includes the bytes used for the FlowSet ID and the length bytes themselves, as well as the combined lengths of any included data records.
Record N - Field N	The remainder of the Version 9 data FlowSet is a collection of field values. The type and length of the fields have been previously defined in the template record referenced by the FlowSet ID/template ID.
Padding	Padding should be inserted to align the end of the FlowSet on a 32 bit boundary. Pay attention that the Length field will include those padding bits.

Table 2.9: NetFlow Version 9 Data FlowSet Field Descriptions [6]

Chapter 3

Kernel modules

What exactly is a kernel module? Modules are pieces of code that can be loaded and unloaded into the kernel upon demand. They extend the functionality of the kernel without the need to reboot the system. For example, one type of module is the device driver, which allows the kernel to access hardware connected to the system. Without modules, we would have to build monolithic kernels and add new functionality directly into the kernel image. Besides having larger kernels, this has the disadvantage of requiring us to rebuild and reboot the kernel every time we want new functionality.

3.1 Programs vs Modules

A program usually begins with a `main()` function, executes a bunch of instructions and terminates upon completion of those instructions. Kernel modules work a bit differently. A module always begin with either the `init_module` or the function you specify with `module_init` call. This is the entry function for modules; it tells the kernel what functionality the module provides and sets up the kernel to run the module's functions when they're needed. Once it does this, entry function returns and the module does nothing until the kernel wants to do something with the code that the module provides. All modules end by calling either `cleanup_module` or the function you specify with the `module_exit` call. This is the exit function for modules; it undoes whatever entry function did. It unregisters the functionality that the entry function registered. Every module must have an entry function and an exit function.

3.2 Building Kernel Module

„kbuild“ is the build system used by the Linux kernel. Modules must use kbuild to stay compatible with changes in the build infrastructure and to pick up the right flags to „gcc.“ Functionality for building modules both in-tree and out-of-tree is provided. The method for building either is similar, and all modules are initially developed and built out-of-tree. The author of an external module should supply a makefile that hides most of the complexity, so one only has to type „make“ to build the module.

3.2.1 How to build

To build external modules, you must have a prebuilt kernel available that contains the configuration and header files used in the build. Also, the kernel must have been built with

modules enabled. If you are using a distribution kernel, there will be a package for the kernel you are running provided by your distribution.

An alternative is to use the „make“ target „modules_prepare.“ This will make sure the kernel contains the information required. The target exists solely as a simple way to prepare a kernel source tree for building external modules.

NOTE: „modules_prepare“ will not build Module.symvers even if CONFIG_MODVERSIONS is set; therefore, a full kernel build needs to be executed to make module versioning work.

3.2.2 Commands syntax

The command to build an external module is:

```
$ make -C <path_to_kernel_src> M=$PWD
```

The kbuild system knows that an external module is being built due to the „M=<dir>“ option given in the command. To build against the running kernel use:

```
$ make -C /lib/modules/‘uname -r‘/build M=$PWD
```

Then to install the module(s) just built, add the target „modules_install“ to the command:

```
$ make -C /lib/modules/‘uname -r‘/build M=$PWD modules_install
```

3.2.3 Options

($KDIR refers to the path of the kernel source directory.)

```
make -C $KDIR M=$PWD
```

-C $KDIR
The directory where the kernel source is located. „make“ will actually change to the specified directory when executing and will change back when finished.

M=$PWD
Informs kbuild that an external module is being built. The value given to „M“ is the absolute path of the directory where the external module (kbuild file) is located.

3.2.4 Targets

When building an external module, only a subset of the „make“ targets are available.

```
make -C $KDIR M=$PWD [target]
```

The default will build the module(s) located in the current directory, so a target does not need to be specified. All output files will also be generated in this directory. No attempts are made to update the kernel source, and it is a precondition that a successful „make“ has been executed for the kernel.

modules
The default target for external modules. It has the same functionality as if no target was specified. See description above.

modules_install

Install the external module(s). The default location is /lib/modules/<kernel_release>/extra/, but a prefix may be added with INSTALL_MOD_PATH (discussed in section 5).

clean
Remove all generated files in the module directory only.

help
List the available targets for external modules.

3.2.5 Creating Kbuild file for external module

In the last section we saw the command to build a module for the running kernel. The module is not actually built, however, because a build file is required. Contained in this file will be the name of the module(s) being built, along with the list of requisite source files. The file may be as simple as a single line:

```
obj-m := <module_name>.o
```

The kbuild system will build <module_name>.o from <module_name>.c, and, after linking, will result in the kernel module <module_name>.ko. The above line can be put in either a „Kbuild“ file or a „Makefile.“ When the module is built from multiple sources, an additional line is needed listing the files:

```
<module_name>-y := <src1>.o <src2>.o ...
```

3.3 Simple example

Consider following simple kernel module **main.c**:

```
#include <linux/kernel.h>
#include <linux/module.h>

MODULE_LICENSE("GPL");

int init_module(void){
        printk(KERN_INFO "Hello world!\n");
        return 0;
}


void cleanup_module(void){
        printk(KERN_INFO "Goodbye world!\n");
}
```

Now let's look at a simple **Makefile** for compiling a kernel module above:

```
obj-m := main.o

all:
        make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
```

```
clean:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

By executing

```
$ make all
```

we get the following output and **main.ko** file is generated:

```
make -C /lib/modules/5.15.0-91-generic/build \\
M=/home/matheas/Desktop/local/module modules
make[1]: Entering directory '/usr/src/linux-headers-5.15.0-91-generic'
CC [M] /home/matheas/Desktop/local/module/main.o
MODPOST /home/matheas/Desktop/local/module/Module.symvers
CC [M] /home/matheas/Desktop/local/module/main.mod.o
LD [M]: /home/matheas/Desktop/local/module/main.ko
BTF [M] /home/matheas/Desktop/local/module/main.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.15.0-91-generic'
```

We can now load ('**insmod**') and unload('**rmmod**') main.ko module by executing respective commands in brackets. We can verify that the main.ko module was successfully loaded and unloaded by going to '**/var/log/syslog**' and we can see the following lines in the log file:

```
Jan 15 16:48:28 matheas-VB kernel: [23363.993115] Hello world!
Jan 15 16:48:32 matheas-VB kernel: [23367.855583] Goodbye world!
```

Chapter 4

Implementation

In this chapter, we will be talking about existing tools, how to add own custom attribute to ipt_netflow probe, and implementation of our own Variance attribute.

4.1 Existing tools

This section describes some of the popular netflow probes, providing a brief introduction to each of them, the features they offer, and their advantages/disadvantages.

4.1.1 nProbe

nProbe, an acronym for NetFlow probe, is an open-source NetFlow v5 probe. The application captures packets flowing on an Ethernet segment, computes NetFlow flows, and exports them to the specified collectors. Users can fully control flow parameters (e.g., flow expiration time) as well as flow collectors.

nProbe's main features include [8]:

- Support for NetFlow v5.
- Ability to keep up with Gbit speeds on Ethernet networks handling without packet sampling on commodity hardware
- Support for major OS, including Unix, Windows, and MacOS X.
- Resource (both CPU and memory): savvy, efficient, designed for environments with limited resources. The source code is available under the GNU GPL license.

Probe has been designed to be small, efficient and easy to embed in hardware. For this reason, the application design is very simple. Packets are captured from the network using libpcap, a portable packet capture library, decoded, and stored on a hash. Each hash bucket contains information about a flow (e.g., total number of packets, flow duration) in order to maintain the flow state. Whenever an expired flow is detected, the corresponding hash bucket is freed, and the flow is emitted. In order to avoid emitting packets containing few flows, the user can specify the minimum number of flows per packet so that the probe does not send the packet until the limit is reached.

4.1.2 Flowmon probe

Flowmon provides a unified network, anomaly, and security analytics solution, leveraging flow and packet data, which helps enable a fast, reliable, and well-secured network for any organization. Keysight Visibility Architecture complements Flowmon by generating enriched flow data (IxFlow) and sending it to Flowmon for analysis. Keysight turns packets into insights via deep packet inspection of application, user devices, and other flow details. For example, IxFlow exports HTTP details not available in traditional Netflow/IPFIX, which Flowmon uses to help organizations identify communication with suspicious or blacklisted domains. The combined solution allows administrators to identify the users, devices, networks and applications that generate undesirable traffic or security threats. Armed with reliable and accurate information, IT can isolate and remove unwanted behaviors. Keysight also shares packet data with Flowmon Probe, which performs network-based Application Performance Monitoring or on-demand full packet capture, as well as allowing data to be shared with customers' other analysis tools, protecting legacy investments.

4.1.3 ipt_netflow probe

High performance NetFlow v5, v9, IPFIX flow data export module for Linux kernel. Created to be useful for linux routers in high-throughput networks. It should be used as iptables target. Unlike nprobe, this ipt_netflow is not a standalone application, but open source kernel module, so we decided to extend ipt-netflow module with variance export feature. Other features of ipt-netflow are [5]:

- High performance and scalability. For highest performance module could be run without conntrack being enabled in kernel. Reported to be able to handle 10Gbit traffic with more than 1500000 pps with negligible server load (on S5500BC).
- NetFlow v5, v9, and IPFIX are fully supported.
- IPv6 option headers, IPv4 options, TCP options, ethernet type, dot1q service and customer VLAN ids, MAC addresses, and
- NAT translations events (from conntrack) using NetFlow Event Logging (NEL). This is standardized way for v9/IPFIXr, but module export such events even for v5 collectors via specially crafted pseudo-records.
- Deterministic (systematic count-based), random and hash Flow Sampling. With appropriate differences in support of v5, v9, and IPFIX.
- SNMP agent (for net-snmp) for remote management and monitoring.
- SNMP-index translation rules, let convert meaningless and unstable interface indexes (ifIndex) to more meaningful numbering scheme.
- Options Templates (v9/IPFIX) let export useful statistical, configurational, and informational records to collector. Such as metering, exporting, sampling stat and reliability stat, sampling configuration, network devices ifName, ifDescr list.
- Module load time and run-time (via sysctl) configuration.
- Flexibility in enabling features via ./configure script. This will let you disable features you don't need, which increase compatibility with custom kernels and performance.

- Easy support for catching mirrored traffic with promisc option. Which is also supporting optional MPLS decapsulation and MPLS-aware NetFlow.

ipt_netflow probe installation CentOS

1. Install kernel sources.

```
# yum install kernel-devel
```

2. Prepare iptables.

```
# yum install iptables-devel
```

3. Prepare net-snmp (optional)

```
# yum install net-snmp net-snmp-devel
```

4. Build and Install

```
~/ipt-netflow# ./configure
~/ipt-netflow# make all install
~/ipt-netflow# depmod
```

ipt_netflow probe running

1. Load kernel module

```
# insmod ipt_NETFLOW.ko destination=127.0.0.1:2055 debug=1
or
# modprobe ipt_NETFLOW destination=127.0.0.1:2055
```

2. Statistics is in /proc/net/stat/ipt_netflow. Dump of all flows is in /proc/net/stat/ipt_netflow_flows

3. You can view parameters and control them via sysctl, example:

```
# sysctl net.netflow
# sysctl net.netflow.hashsize=32768
```

4. Example of directing all IPv4 traffic into the module:

```
# iptables -I FORWARD -j NETFLOW
# iptables -I INPUT -j NETFLOW
# iptables -I OUTPUT -j NETFLOW
```

5. If you want to account IPv6 traffic you should use protocol 9 or 10. Example of directing all IPv6 traffic into the module:

```
# sysctl net.netflow.protocol=10
# ip6tables -I FORWARD -j NETFLOW
# ip6tables -I INPUT -j NETFLOW
# ip6tables -I OUTPUT -j NETFLOW
```

4.2 Add custom attribute to ipt_netflow probe

In this part, we examine the ipt_netflow probe's source code and explain how to add a custom attribute of our own to the netflow v9 export packet. You can find out exactly which files, functions, structures, and other elements you need to change in order to get the custom attribute to be exported by ipt_netflow.

4.2.1 Header file

Let's start with header file modifications. When we want to add a new attribute to export, we need to begin by modifying two things: structure `ipt_netflow_tuple`, which contains various elements that are collected from packet flow (number of packets, number of bytes, ...). Then we need to define export Field type number and Field Length by adding a new member to `#define Elements` section.

- one(Field Type, v9_name (UPPERCASED), Field Length) - defined attribute will be only exported as part of netflow v9 records.
- two(Field Type, v9_name (UPPERCASED), ipfix_name (CamelCased), Field Length) - defined attribute will be exported as part of both netflow v9 and IPFIX records.

4.2.2 Source code

For every new member in `ipt_netflow_tuple`, we need to modify `netflow_target` function. This function takes multiple parameters, including skbuff, xt_target, net_device parameters, therefore we have available every kernel information about packet, input/output interface, etc.

We need to define Data Template by defining macro as shown in code:

```
#define BTPL_{NAME} mask //mask is a value representing bit set to 1
    that is not already used by another data templates
```

Now that we have defined macro for our attribute, we need to enable export for this attributeby modifying function `netflow_export_flow_tpl` and adding the following line:

```
tpl_mask |= BTPL_{NAME};
```

Then we need to define the (type, length) template structure variable for out attribute. There is already defined data type `base_template`, so all we need to do is declare a global static variable as shown in the code snippet below, where v9_name is the name defined in the Elements section of the header file.

```
static struct base_template template_{name} = {
    .types = {
           v9_name,
           0
    }
};
```

And add this template variable to list of templates by adding a following snippet of code to `get_template` function.

```
if (tmask & BTPL_{NAME}){
    tlist[tnum++] = &template_{name};
}
```

At this moment we have defined and implemented everything for data template, now we need to export the attribute value itself. We can do it by modifying `switch` in function `add_tpl_field` utilizing functions `put_unaligned/put_unaligned_be{8,16,32,64}`.

```
case v9_name:
    put_unaligned_be64({our_value/variable}, ptr);
    break;
```

4.3 Variance implementation

Variance feature is used in detection of DNS of HTTPS attacks. Variance feature for the flow can be calculated with the formula 4.1 from Cisco website [2], where:

- **cbs** - number of bytes squared
- **pkts** - number of packets
- **bytes** - number of bytes

$$Variance = (cbs/pkts) - (bytes/pkts)^2 \tag{4.1}$$

4.3.1 Header file

We extend the `ipt_netflow` structure with `nr_bytes_squared` member to store total bytes squared value for the Variance calculation.

```
u_int64_t nr_bytes_squared;
```

We add a new element `CISCO_VARIANCE` to **Elements** defined in ipt_netflow.h. Where value 37 represents Field Type in Table 2.7 and value 4 represents Field Length in 2.7.

```
#define Elements \
    .
    .
    .
    two(37, CISCO_VARIANCE, ciscoVariance, 4) \
    .
    .
    .
```

4.3.2 Source file

Now that we have defined a `nr_bytes_squared` variable in header file, we need to modify `netflow_target` function and add new line:

```
nf->nr_bytes_squared += pkt_len*pkt_len;
```

We define new `BTPL_VARIANCE` data template option with value 0x02000000 (first unused bit in sequence from least significant bit).

```
#define BTPL_VARIANCE 0x02000000
```

Then in function `netflow_export_flow_tpl` we have to enable export of variance by ORing `BTPL_VARIANCE` to `tpl_mask` variable.

```
tpl_mask |= BTPL_VARIANCE
```

In the global scope of the source file, we define the template `template_variance` variable for variance export.

```
static struct base_template template_variance = {
    .types = {
        CISCO_VARIANCE,
        0
    }
};
```

Then we need to add this template to switch in function `get_template()`:

```
if(tmask & BTPL_VARIANCE){
    tlist[num++] = &template_variance;
}
```

And finally in function `add_tpl_field` we do calculation of variance and add the result to export packet.

```
case CISCO_VARIANCE:
    put_unaligned_be32((u_int32_t)(nf->nr_bytes_squared/nf->nr_packets) -
        ((nf->nr_bytes/nf->nr_packets)*(nf->nr_bytes/nf->nr_packets)));
```

Chapter 5

Testing

For testing purposes to verify that variance attribute is calculated correctly, we used following tools:

- ping
- tshark
- nfdump

Ping tool is used for traffic generation, main feature of this tool is option to specify size of sent packet. nfdump is used as NetFlow collector (hooked on 127.0.0.1:2555), however Variance attribute is cisco proprietary and nfdump does not have an option to collect that attribute, we need to set Variance attribute to known ID (sourceAS). Tshark is for capturing export of flows.

5.1 First sample

For first sample we send 8 packets with byte sizes [64, 2048, 128, 256, 32, 1024, 128, 128]. Ping sends packets with size+8, therefore for values [72, 2056, 136, 264, 40, 1032, 136, 136] we can calculate expected variance as follows:

- Number of packets (pks): 8
- Total byte size (bytes): 3872
- Total byte size squared (cbs): 5424128
- Expected Variance:

$$(cbs/pkts) - (bytes/pkts)^2 = Variance$$

$$(5424128/8) - (3872/8)^2 = 443760$$

Now we need to get the actual result:

1. Execute following pings:

```
# ping -c 1 -s 64 8.8.8.8 >/dev/null
# ping -c 1 -s 2048 8.8.8.8 >/dev/null
# ping -c 1 -s 128 8.8.8.8 >/dev/null
# ping -c 1 -s 256 8.8.8.8 >/dev/null
# ping -c 1 -s 32 8.8.8.8 >/dev/null
# ping -c 1 -s 1024 8.8.8.8 >/dev/null
# ping -c 1 -s 128 8.8.8.8 >/dev/null
# ping -c 1 -s 128 8.8.8.8 >/dev/null
```

2. Verify actual result in „/proc/net/stat/ipt_netflow_flows:

```
[matheas@localhost Desktop]$ cat /proc/net/stat/ipt_netflow_flows
# hash a dev:i,o proto src:ip,port dst:ip,port nexthop tos,
   tcpflags,options,tcpoptions packets bytes ts:first,last
   variance
1 2320f 0 -1,1 17 127.0.0.1,56463 127.0.0.1,2055 0.0.0.0 0,0,0,0 5
    1360 1463,1463 0
2 6aa66 0 2,-1 1 8.8.8.8,0 10.0.2.15,0 0.0.0.0 0,0,0,0 35 3160
   12055,1659 206
3 4dfab 0 1,-1 17 127.0.0.1,56463 127.0.0.1,2055 0.0.0.0 0,0,0,0 5
    1360 1463,1463 0
4 f4ba 0 -1,2 1 10.0.2.15,0 8.8.8.8,2048 10.0.2.2 0,0,0,0 40 20160
    12121,1732 443760
```

3. Verify actual result in generated pcap file (screenshot from WireShark 5.1)

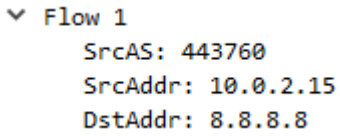

Figure 5.1: Screenshot of Sample 1

5.2 Second Sample

For second sample we send 15 packets with same size [2048, 2048, 2048, 2048, 2048, 2048, 2048, 2048, 2048, 2048, 2048, 2048, 2048, 2048, 2048] . Again Ping sends packets with size+8, therefore for values [2056, 2056, 2056, 2056, 2056, 2056, 2056, 2056, 2056, 2056, 2056, 2056, 2056, 2056, 2056] we can calculate expected variance as follows:

- Number of packets (pks): 15
- Total byte size (bytes): 30840
- Total byte size squared (cbs): 63407040
- Expected Variance:

$$(cbs/pkts) - (bytes/pkts)^2 = Variance$$

$$(63407040/15) - (30840/15)^2 = 0$$

Now we need to get the actual result:

1. Execute following pings:

```
# ping -c 1 -s 2048 8.8.8.8 >/dev/null
# ping -c 1 -s 2048 8.8.8.8 >/dev/null
# ping -c 1 -s 2048 8.8.8.8 >/dev/null
# ping -c 1 -s 2048 8.8.8.8 >/dev/null
# ping -c 1 -s 2048 8.8.8.8 >/dev/null
# ping -c 1 -s 2048 8.8.8.8 >/dev/null
# ping -c 1 -s 2048 8.8.8.8 >/dev/null
# ping -c 1 -s 2048 8.8.8.8 >/dev/null
# ping -c 1 -s 2048 8.8.8.8 >/dev/null
# ping -c 1 -s 2048 8.8.8.8 >/dev/null
# ping -c 1 -s 2048 8.8.8.8 >/dev/null
# ping -c 1 -s 2048 8.8.8.8 >/dev/null
# ping -c 1 -s 2048 8.8.8.8 >/dev/null
# ping -c 1 -s 2048 8.8.8.8 >/dev/null
# ping -c 1 -s 2048 8.8.8.8 >/dev/null
```

2. Verify actual result in "/proc/net/stat/ipt_netflow_flows:

```
[matheas@localhost Desktop]$ cat /proc/net/stat/ipt_netflow_flows
# hash a dev:i,o proto src:ip,port dst:ip,port nexthop tos,
   tcpflags,options,tcpoptions packets bytes ts:first,last
   variance
1 6aa66 0 2,-1 1 8.8.8.8,0 10.0.2.15,0 0.0.0.0 0,0,0,0 91 8736
   23823,3070 0
2 f4ba 0 -1,2 1 10.0.2.15,0 8.8.8.8,2048 10.0.2.2 0,0,0,0 105
   110460 23870,3126 0
```

3. Verify actual result in generated pcap file (screenshot from WireShark 5.2)

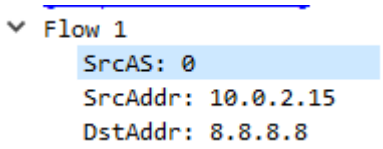

Figure 5.2: Screenshot of Sample 2

Chapter 6

Conclusion

The first portion of Chapter 2 provides a brief introduction to network monitoring and packet flow capture. It outlines the procedure for gathering and examining data packets as they travel over a network. Subsequently, the second section presents Netflow, a network protocol designed for network traffic monitoring by Cisco Systems. In-depth information about Netflow's architecture and how its parts work together to produce packet flow records is covered in the third section. The fourth portion then moves on to talk about Netflow version 5, emphasizing its features and traffic analysis capabilities. The fifth and final section examines Netflow version 9, highlighting how it uses templates to provide extensible and adaptable flow monitoring in intricate network contexts.

An overview of kernel modules is given in the first part of the Chapter 3, which also compares and contrasts them with conventional programs and explains their function in the operating system. The purpose of this section is to make the differences between user-space applications and kernel modules clear, highlighting the functions and interactions of each. After that, the second section walks readers through the process of creating a kernel module by going over the crucial actions and factors to take into account when creating and incorporating unique kernel functionality. The main emphasis is on useful advice and recommended procedures for developing kernel modules that satisfy particular system specifications.

The first part of the Chapter 4 gives a thorough description of the variance computation process, including the mathematical foundations and the importance of variance in statistical analysis. To help readers comprehend, this section provides examples and step-by-step directions. Subsequently, the second section presents popular tools for network traffic monitoring and analysis, including nprobe, flowmon, and ipt_netflow. To help readers choose the best solution for their unique needs, each tool's characteristics, capabilities, and use cases are covered. In the context of network traffic analysis, the third and final part describes in detail the changes made to the source and header files in order to execute variance computation. This section provides information on the code modifications needed to include variance calculation. This section provides information on the code modifications needed to build custom solutions suited to specific analytical requirements or integrate variance calculation capability into currently available tools.

The emphasis of the last Chapter 5 is on verifying the accuracy and dependability of the variance computation capability that has been implemented. An introduction of the testing process is given at the beginning of the chapter, with a focus on the value of employing a variety of datasets and strict validation procedures.

Bibliography

[1] Jun 2011. Available at: https://www.cisco.com/en/US/technologies/tk648/tk362/technologies_white_paper09186a00800a3db9.html.

[2] Cisco, Oct 2013. Available at: https://www.cisco.com/c/en/us/td/docs/ios/fnetflow/command/reference/fnf_book/fnf_01.html.

[3] IBM, Feb 2022. Available at: https://www.ibm.com/docs/en/npi/1.3.1?topic=insight-netflow-v9-formats.

[4] IBM, Mar 2024. Available at: https://www.ibm.com/topics/netflow.

[5] AABC. *Ipt-netflow* [https://github.com/aabc/ipt-netflow]. GitHub, 2021.

[6] CLAISE, B. *Cisco Systems NetFlow Services Export Version 9* [Internet Requests for Comments]. RFC 3954. RFC Editor, October 2004. http://www.rfc-editor.org/rfc/rfc3954.txt. Available at: http://www.rfc-editor.org/rfc/rfc3954.txt.

[7] CLAISE, B., TRAMMELL, B. and AITKEN, P. *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information* [Internet Requests for Comments]. STD 77. RFC Editor, September 2013. http://www.rfc-editor.org/rfc/rfc7011.txt. Available at: http://www.rfc-editor.org/rfc/rfc7011.txt.

[8] DERI, L. NProbe: an Open Source NetFlow Probe for Gigabit Networks. january 2003.

[9] DUFFIELD, N. Sampling for Passive Internet Measurement: A Review. *Statistical Science*. august 2004, vol. 19. DOI: 10.1214/088342304000000206.

[10] HOFSTEDE, R., CELEDA, P., TRAMMELL, B., DRAGO, I., SADRE, R. et al. Flow monitoring explained: from packet capture to data analysis with NetFlow and IPFIX. *IEEE communications surveys & tutorials*. IEEE. november 2014, vol. 16, no. 4, p. 2037–2064. DOI: 10.1109/COMST.2014.2321898. ISSN 1553-877X. eemcs-eprint-25179.

[11] MICHEEL, J., DONNELLY, S. and GRAHAM, I. Precision timestepping of network packets. january 2001, p. 273–277. DOI: 10.1145/505202.505236.

[12] ORGERIE, A.-C., GONÇALVES, P., IMBERT, M., RIDOUX, J. and VEITCH, D. Survey of Network Metrology Platforms. *Proceedings - 2012 IEEE/IPSJ 12th International Symposium on Applications and the Internet, SAINT 2012*. july 2012, p. 220–225. DOI: 10.1109/SAINT.2012.41.

[13] Santos, O. and Muniz, J. *CCNA Cyber Ops SECOPS 210-255 Official Cert Guide*. 1stth ed. Cisco Press, 2017. ISBN 1587147033.

[14] Vormayr, G., Fabini, J. and Zseby, T. Why are My Flows Different? A Tutorial on Flow Exporters. *IEEE Communications Surveys & Tutorials*. 2020, vol. 22, no. 3, p. 2064–2103. DOI: 10.1109/COMST.2020.2989695.

[15] Zseby, T., Molina, M., Duffield, N., Niccolini, S. and Raspall, F. *Sampling and Filtering Techniques for IP Packet Selection* [Internet Requests for Comments]. RFC 5475. RFC Editor, March 2009.