Bachelor's thesis

Information and Communications Technology

2025

Palmer Morten

Recording Packet Drop Exceptions in a Service Provider Network with Juniper Resiliency Interface

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

Bachelor’s Thesis | Abstract

Turku University of Applied Sciences

Information and Communications Technology

2025 | 59 pages

Palmer Morten

Recording packet drop exceptions in service provider network with Juniper Resiliency Interface

Packet drops occur frequently in networks and often disrupt the services that rely on them. Developing methods to detect and mitigate packet loss can improve fault recovery time and enhance customer satisfaction. This thesis examined a new feature from Juniper Networks enables the recording of packet drops on Juniper routers.

Although the feature is a new technology, it relies on established technologies, such as IPFIX, which is used to gather flow data from network traffic. The Juniper Resiliency Interface, instead of collecting information from network traffic, gathers data from dropped packets, revealing details such as the time, location, and reason for the packet drop.

In addition to the theoretical study, the thesis focused on building a system that collects, processes, stores, and visualizes the packet drop data. The core of the system is a Python program, which collects, parses, and saves the data while also managing the potential exceptions produced by networks. For visualization and efficient data storage, the system utilizes the Grafana platform and Timescale DB.

The thesis work was conducted in a lab environment, where the feature proved to provide both accurate and useful data about packet drops. The most valuable result from the thesis was the visualization of the packet drop data, which enables the identification of potential fault points in the network.

Keywords: Networks, Data bases, Python, Juniper Networks, IPFIX, Flow monitoring.

Opinnäytetyö (AMK) / Tiivistelmä

Turun ammattikorkeakoulu

Tieto- ja Viestintätekniikka

2025 | 59 sivua

Palmer Morten

Verkkopakettien pudotusten nauhoittaminen internet-palveluntarjoajan verkossa Juniper Resiliency Interface ominaisuuden avulla

Verkkopakettien pudotuksia tapahtuu verkossa jatkuvasti, ja ne vaikuttavat usein negatiivisesti verkossa toimiviin palveluihin. Kehittämällä menetelmiä pakettipudotusten ymmärtämiseen ja vähentämiseen voidaan parantaa vikojen korjausaikaa ja siten lisätä asiakastyytyväisyyttä. Opinnäytetyössä tutustuttiin Juniper Resiliency Interface -ominaisuuteen, joka mahdollistaa verkkopaketti pudotusten nauhoittamisen Juniperin laitteilla.

Vaikka ominaisuus on suhteellisen uusi, perustuu se jo tunnettuihin teknologioihin, kuten IPFIX -protokollaan, jolla on mahdollista kerätä dataa verkkoliikenteestä. Verkkoliikenteen sijaan Juniper Resiliency Interface kerää dataa ainoastaan verkkopaketti pudotuksista, paljastaen esimerkiksi ajan, sijainnin ja syyn, miksi verkkopaketti on jouduttu pudottamaan.

Teoreettisen tarkastelun lisäksi opinnäytetyössä rakennettiin järjestelmä, jolla verkkopaketti pudotus dataa kerätään, käsitellään ja visualisoidaan. Tämän järjestelmän ytimessä toimii Python -ohjelma, joka kerää, käsittelee ja tallentaa tiedot sekä hallitsee verkoista johtuvia mahdollisia poikkeustilanteita. Visualisointiin ja tehokkaaseen tietojen tallennukseen järjestelmä käyttää Grafana-alustaa ja TimescaleDB-tietokantaa.

Opinnäytetyö toteutettiin laboratorioympäristössä, jossa ominaisuuden huomattiin tuottavan tarkkaa ja hyödyllistä tietoa. Työn merkittävin tulos oli verkkopaketti pudotusten visualisointi, joka mahdollistaa verkon vikakohtien paikantamisen.

Asiasanat: Tietoverkot, Tietokannat, Python, Juniper Networks, IPFIX, Flow monitoring.

Contents

Code

Figures

Tables

List of abbreviations

CPU	Central Processing Unit
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name Service
DSCP	Differentiated Services Code Point
DTLS	Datagram Transport Layer Security
FPC	Flexible PIC concentrator
IANA	Internet Assigned Numbers Authority
IETF	Internet Engineering Taskforce
IPFIX	IP Flow Information Export
JRI	Juniper Resiliency Interface
PFE	Packet Forwarding Engine
PSQL	PostgreSQL
RE	Routing engine
SCP	Secure Copy Protocol.
SCTP	Stream Control Transmission Protocol
SSH	Secure Shell
RFC	Request for Comments
TLS	Transport Layer Security
TTL	Time To Live
UDP	User Datagram Protocol

1 Introduction

Packet drops occur in networks due to varying reasons. These include errors in transmission, firewall filters, network congestion, expiring time-to-live counters, among others [1]. When packets are dropped, it most often affects the services traversing over the network. Detecting and understanding the packet drops in networks is a challenging task even for an experienced network engineer. In many cases, issues are only noticed after they have already surpassed.

Traditional methods for understanding and detecting packet drops include applying firewall filters, capturing traffic, and analyzing data from multiple points in networks. More specialized methods include using network probes, monitoring services and path analysis methods.[2] [3] In recent years, even machine learning algorithms have been employed to detect and mitigate packet loss. For example, in [4], a Random Forest algorithm was used to identify packet loss from network flow data, achieving a detection accuracy of up to 93%.

As networks and their capacity have expanded, identifying and applying new methods for detecting packet drops is needed. Although networking devices have evolved, meaning that the congestion and software bugs resulting in packet drops are not as prevalent as before, there still is packet loss happening. As stated in [4], “The current model of providing Quality of Service works well within a network but does not scale in multi-provider environments. This may be exhibited as packet loss occurring, for example, at peering points.”

Juniper Networks recently released a feature called Juniper Resiliency Interface. The feature allows the users to record packet drop occurrences on Juniper Networking devices and save the packet drop information to an external or a local collector. This feature utilizes the fact that the network devices always understand why the packet has to be dropped. The packet drop reports provide users with information such as, packet drop reasons, packet drop time, flow direction, interfaces and more.[1]

The aim for the thesis is to explore the Juniper Resiliency Interface feature and to explore options, by which this feature can be implemented into a network. The objective is to build a self-sufficient system, which records, collects, processes, and visualizes the packet drop data.

The thesis was commissioned by the Finnish Internet Service Provider DNA. The feature is believed to benefit the engineers in fault identification. Alternatively, the product could be used for providing more detailed information to the customers, thus improving customer satisfaction. For safety reasons critical information such as IP addresses, exact configurations, VLANs, usernames, passwords, safety details and such will not be revealed. They will, however, be explained or replaced by adequate information. For example, public IP addresses will be replaced with private IP addresses, or direct safety configuration will be explained rather, than directly shown.

The thesis is composed of five chapters. The first chapter is the introduction. The second chapter studies the theoretical requirements of the feature and the technologies on which the feature is based. The third chapter will demonstrate the designed system architecture and technologies used. The fourth chapter introduces the key results achieved from the system tests and the fifth chapter focuses on concluding the thesis topic and discussing the future work.

This thesis assumes that the reader has some degree of understanding of topics, such as networking, Python programming, relational databases, containerization, and data visualization. These topics will be explained in the context of their application within the implementation, rather than through in-depth explanations of how they function.

AI-assisted proofreading tools were utilized during the writing process of this thesis to support grammar checking.

2 Juniper Resiliency Interface

Juniper Resiliency Interface is a Juniper Networks proprietary product developed for recording packet drop occurrences in Juniper networking devices. Its purpose is to collect reports of packet drop occurrences and to send those reports to either a local or an external collector. The records of the packet drops are transported in an IPFIX packet format, which provides the user with information, such as packet drop reason, ingress and egress interfaces, occurrence time, among others. [1]

Juniper Resiliency Interface as a process works as follows: Network packets are examined at the forwarding plane of the router. If the packets contain errors and they cannot be forwarded, the router drops the packet and sends a report of the packet drop to the predefined collector. [5] Figure 1 illustrates the JRI (Juniper Resiliency Interface) process working on a Juniper router. The packet stream can be seen as the blue and red packets arriving at the router. The JRI process then captures the red dropped packets and sends a report of them.

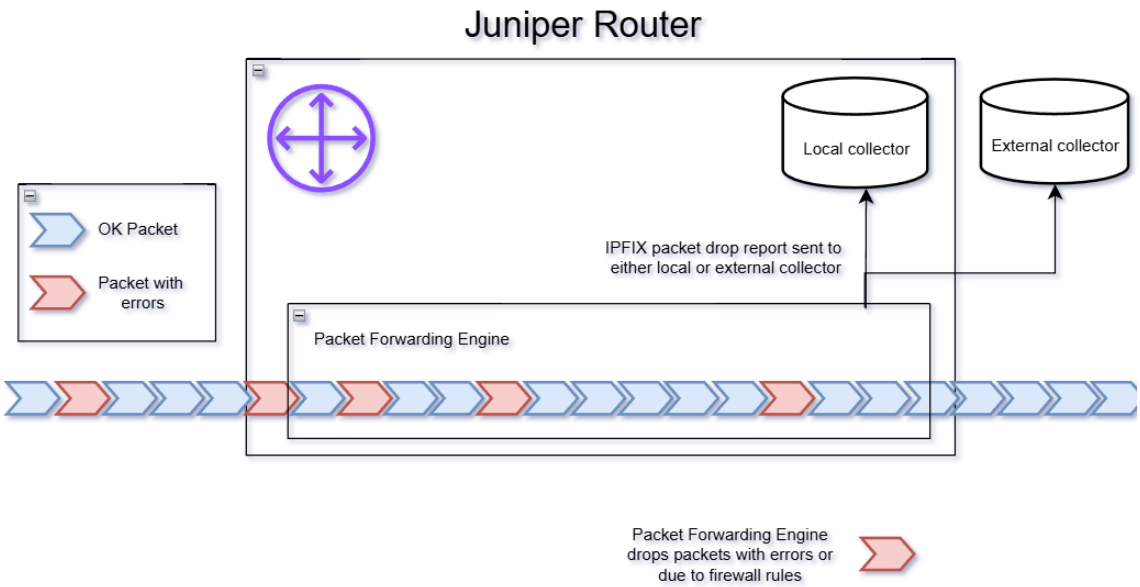


Figure 1. Illustration of the JRI process. Figure illustrates the process as illustrated by Julian Lucek and Chaitanya Munukutla [5]

Juniper Resiliency Interface also offers the possibility of collecting exception data from the routing protocol process and Junos kernel. These exceptions, instead of IPFIX are reported with streaming telemetry.[1] As for the thesis work, the focus will be solely on the packet drop exceptions.

Although the main focus of the thesis will be on Juniper Resiliency Interface and the implementation of it, it will cover some topics and technologies, which the Juniper Resiliency Interface feature is based on. These features include Flow monitoring as a process, IPFIX protocol and Juniper Inline monitoring services.

2.1 Flow Monitoring

Flow monitoring is defined as a process, in which information from network traffic is captured from a networking device by analyzing the packet headers from the traffic with predefined information fields. Flow monitoring also often includes the use of an external collector, to which this monitored flow information is collected for post processing.[6]

Alternatively flow monitoring can be described as a technology, which captures information from network traffic by grouping several packets into flows. Flows can be defined as a series of network traffic owning common characteristics, such as protocol, time, source- and destination addresses.[7]

Figure 2 illustrates the flow monitoring process. It demonstrates the router observing three different flows, illustrated on the figure with different colors. It then processes and sends this information to a collector.

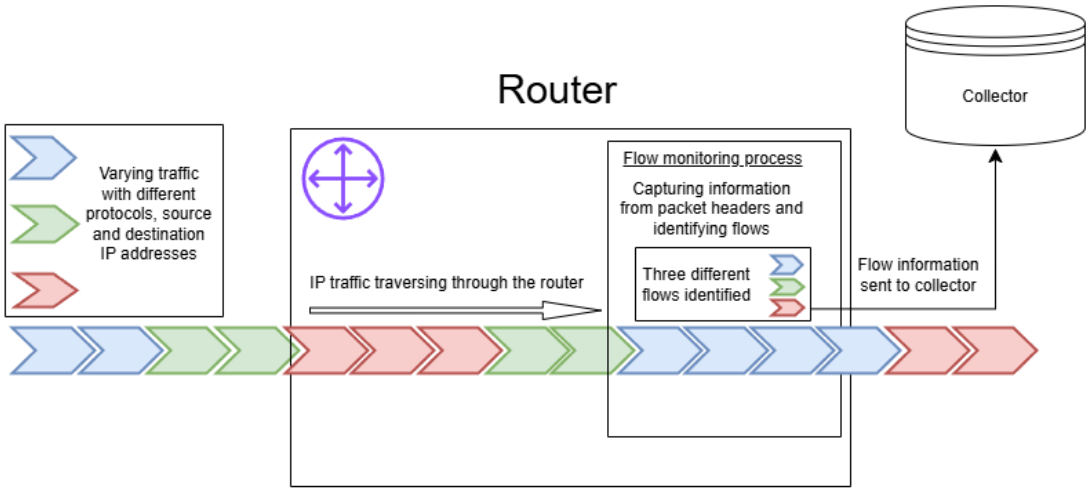


Figure 2. Illustration of the flow monitoring process.

Flow monitoring as a process can be essentially divided into four core phases. The initial phase for flow monitoring starts with capturing the flow information from observation points and pre-processing it .The observation points are often either the line cards or the interfaces of the networking device. The second phase of the flow monitoring is called the metering and the exporting phase. This phase consist of grouping the observed traffic as flows, by common characteristics and exporting this collected flow information to a collector point. Third phase for the process is defined as a collector phase. This phase conducts the collection, storing and pre-processing the flow information exported by the networking device. Fourth and the final phase is the analysis. It can be described as a process where this collected information can be used for further implementation. [7]

In contrast to traditional packet capture, where all the packets and its contents are captured flow monitoring only captures certain information from the packet headers. It is, thus, a lot less intensive and can be offered as a more scalable and cost-effective solution for continuous network traffic monitoring.

Use cases for flow monitoring can be found both in the service provider networks as well as in the enterprise networks.[6] It is widely used for collecting network forensic and billing information, which can then be used for enforcing peering agreements, customer billing and security monitoring. Additionally, it

can be used for creating network baselines and ensuring the collection of information required by the law.

2.2 IPFIX

Juniper Resiliency Interface is not a direct protocol, but rather a technology, which makes use of other protocols. Actual protocol implementation, which is used by the Juniper Resiliency Interface is the use of the IPFIX protocol. JRI uses it for exporting the packet drop information from the device. [1]

IPFIX is an IETF protocol designed to standardize the export process of the IP traffic flow information in networking devices.[8]. IPFIX was developed by the IPFIX working group in order to provide a vendor neutral and standardized protocol to compensate the Cisco's NetFlow v9 product. The protocol was outlined originally in the RFC 3917 and in its current format it is standardized in the RFC 7011. [8]

2.2.1 IPFIX Architecture and Components

As a standardized implementation of the flow monitoring export process and an extension of the NetFlow v9 protocol, IPFIX follows the typical architecture for flow monitoring technologies. Implementation for IPFIX is composed of the same core processes as described earlier. However, it does have some components, which are not standardized for all of the other flow monitoring technologies. [9]

The initial phase for the IPFIX is the capturing of traffic flow from the observation points. Traffic flow as described in RFC 7011 is a group of IP packets or frames recorded at a certain point in the network at a certain time, owning other common characteristics such as destination, source, and protocol.[8]

Observation point is a point in the network where the flow can be recorded from. This is typically an interface or a line card of a router or a switch. IPFIX also allows for extension of the observation point component, to allow the classification of observation domains. Observation domains are a greater set of observation points. An example of an observation domain can be a router, switch, line card, which is composed of multiple other observation points, such as interfaces. [8] Observation domains allow the configuration of observation domain identifiers, which are transported as a part of the IPFIX packet. This in turn helps in differentiating varying flow streams arriving at the collector. [10]

The second phase for the IPFIX is the metering process. The process takes as input the information recorded during the flow capturing at the observation points and processes the flow information, by predefined parameters. [8] For metering and processing IPFIX uses predefined templates and information fields.

The processes following the metering process are the exporting- and the collection processes. These processes transport the collected information from the exporting devices to the collecting devices. An example of these can be a router as an exporting device sending traffic to an external collector. [8]

The difference of IPFIX compared to other similar products, such as NetFlow v9, are in its protocol definitions. For IPFIX implementations SCTP transport protocol support is a mandatory requirement. Another separation from the Netflow v9 is the requirement of a mechanism of data encryption for IPFIX traffic. This is conducted with protocols such as DTLS and TLS. [8]

2.2.2 IPFIX information elements and templates

The most notable component of the IPFIX is the use of templates and information elements [11]. They allow for customization of the information carried by the IPFIX protocol. Template is a combination of type and length pairs of information elements used structuring the IPFIX packet.[8] It can be considered as a rule, describing the following fields in an IPFIX packet. For

templates it is useful to use a unique template id, so the conversating IPFIX devices can differentiate between the templates and the fields described by the templates. [12]

Information elements are the specific IPFIX attribute types described in the template.[8] Task for information elements is to describe the information carried by the IPFIX packet and to describe the datatype carried as an information element. [11]

An illustration of an IPFIX packet is shown in Figure 3. In its contents it describes the Template ID and the layout of the information elements within.

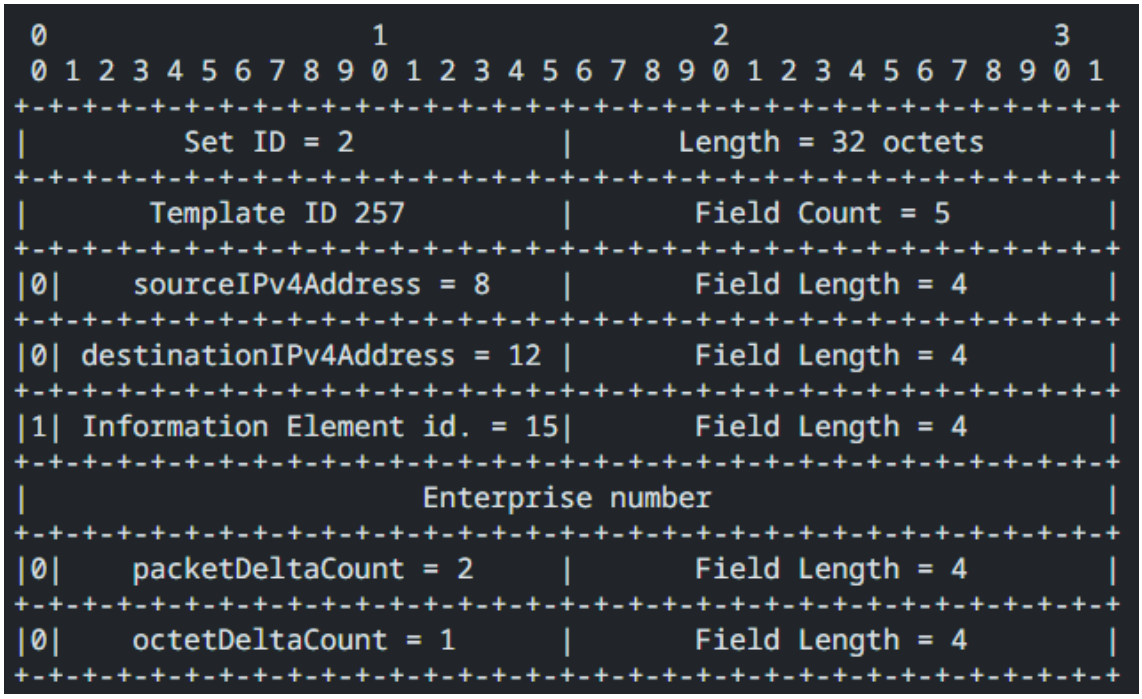


Figure 3. Example of the IPFIX packet structure. [8]

All of the information elements, as of now, are described by the IANA. In its current form, IANA holds records of 529 varying information elements. The information elements from 1-127 are compatible with NetFlow v9 protocol and the rest are IPFIX specific information elements [13].

In Table 1 are examples of the IPFIX information elements. The first three of the examples are compatible with the NetFlow v9.

Table 1. Example of IPFIX information elements.[14]

ElementID	Name	Abstract Data Type	Example
11	destinationTransportPort	unsigned16	80
56	sourceMacAddress	macAddress	aa:bb:cc:11:22:33
82	interfaceName	string	gigabitEthernet1/1
403	originalExporterIPv4Address	ipv4Address	10.10.10.10

The IPFIX protocol also allows the use of enterprise specific information elements. This consolidates the production of non-standardized products, as organizations can develop their own information elements and data that can be carried by IPFIX. Use of these require the use of enterprise bit. Enterprise bit allows the differentiation between information elements specified by IANA and the enterprise. [8]

2.2.3 IPFIX and Juniper Resiliency Interface

Juniper resiliency interface uses IPFIX as a technology for exporting the packet drop information. [1] It must be informed that although it uses IPFIX, it is not a pure implementation of the IPFIX. As IPFIX demands the possibility of the SCTP protocol for transport, JRI only offers the use of UDP protocol.[1] It does however follow IPFIX, with the use of templates, information elements and the process structure. Information about the packet drops is exported as an IPFIX packet and all the information related to the packet drops is composed of the IPFIX information elements. [15]

The Figure 4 illustrates an IPFIX packet produced by the JRI. It portrays a packet drop where the packets TTL has expired, and the router has had to drop it. This information is relayed in the exceptionCode information field. The other information fields are nhIndex, oifIndex, underlyingiifIndex, iifIndex, flowDirection, datalinkFrameSize and datalinkFrameSection. The index values

represent the physical indexes on the device, such as interfaces and sub interfaces.

```
***** Netflow/IPFIX ****
Version: 10
Length: 106
Export time: 1695116517
Seq no: 3397
ObservationDomainID: 65536
setID: 1024
setLength: 90
field_type: exceptionCode, field_length: 2
value: TTL Expired
field_type: nhIndex, field_length: 4
value: 707
field_type: oifIndex, field_length: 4
value: 0
field_type: underlyingiifIndex, field_length: 4
value: 0
field_type: iifIndex, field_length: 4
value: 362
field_type: flowDirection, field_length: 1
value: 00
field_type: dataLinkFrameSize, field_length: 2
value: 1446
field_type: dataLinkFrameSection, field_length: 64
value: 2c6bf561 482a2c6b f5e6d72a 88470002
31014500 05948050 00000101 b4480b65
69010b00 006b0800 d64b023a 00006509
6ce50007 b6370809 0a0b0c0d 0e0f1011
```

Figure 4. Example of a IPFIX packet from a packet drop exception.

As pre-existing IPFIX information elements did not cover all the requirements for the JRI. An IETF draft requesting new IPFIX information elements was co-authored by Juniper Engineers. [5] The requested new IPFIX information elements include forwardingStatusCode, forwardingNextHopID, forwardingLookupType and underlyingIngressInterface These IPFIX elements are not portrayed in the example packet above and the draft in which they were introduced has not passed as of now. [14] This does however imply that the information exported by the JRI might be subjectable to change.

2.3 Inline Monitoring Services

Inline Monitoring Services is a Juniper Networks products developed as an enhancement for flow monitoring. It allows service providers to have more visibility into traffic flows, by allowing flow monitoring process to be split into multiple instances on the device. These instances can then be configured for different sampling requirements. Instances could for example be configured to have differing interfaces or sampling rates, for differing flows. In Junos OS, the Inline Monitoring Services allow the configuration of up to sixteen different instances, while Junos OS Evolved only supports seven. [16]

It also eases the flow monitoring process on the networking devices. Normal flow monitoring process would require the network element to provide the packet parsing and aggregation. The inline monitoring service instead alleviates this by only sampling the packets, adding some metadata, and sending it to a collector for further use. [16]

For the thesis work, the configuration of the JRI is done as part of the inline-monitoring instances.

2.4 Juniper Resiliency Interface Components and Configuration

The Juniper Resiliency Interface as a technology works similarly to flow monitoring processes, it also extends the use of inline monitoring services on top of the metering and exporting processes, allowing for customized instances of packet drop recording.[1] The configuration does not completely follow the flow monitoring process structure, but can be divided into parts that mimic it. For the simplicity, this thesis will explain the configuration and the components in the order, as they would be involved in the packet drop capturing process. This way the components can also be divided into groups that resemble the flow monitoring architecture.

It is important to note that Juniper routers can be divided into two main groups by their operating system. Junos OS, which runs on top of FreeBSD kernel and

Junos OS Evolved, which is built on top of Linux kernel. Although the newer Junos OS Evolved has been developed to be similar to the Junos OS, there are differences to the Junos OS.[17] Some of the differences also extend to the configuration and the components of the Juniper Resiliency Interfaces. The differences will be explained during their respective chapters.

2.4.1 Enabling the Juniper resiliency interface process

For enabling the JRI process, it is required to configure the system wide resiliency exception setting. In Junos OS, exception capturing can be configured for forwarding, routing, and operating system exceptions. The Code 1 demonstrates how to enable this function on the Junos OS. For Junos OS evolved, it is only possible to capture forwarding and routing errors. For Junos OS, the exception capturing process can be enabled, as per the example below. Here the system configured to capture all of the possible exceptions. [1]

Code 1. Enabling the exception recording on Junos OS

```
set system resiliency exceptions forwarding
set system resiliency exceptions routing
set system resiliency exceptions os
```

For Junos OS Evolved the configuration works by enabling only the forwarding and routing exceptions option.[1] Code 2 demonstrates how this is configured on Junos OS Evolved devices.

Code 2. Enabling the exception recording on Junos OS EVO

```
set system resiliency exceptions forwarding
set system resiliency exceptions routing
```

2.4.2 Observation Points

As packets arrive into the router, its header is inspected in order to decide where the packet should be forwarded. This function works at the data plane of the router. In Juniper routers this section of the router is combined of components, such as interfaces, line cards, forwarding table, application-specific integrated circuits, micro kernel and the chassis processes. In juniper terms this combination of components is called the Packet Forwarding Engine. [18] Its main task is to perform the header look up and decide the forwarding action for the packets. This is also the section of the router, where the packets are dropped due to varying reasons.

For configuring the observation points on Junos OS devices works by configuring the exception reporting, the varying exceptions categories and the inline monitoring instance on the line cards called the FPCs and the PFEs. In Code 3 exception reporting is applied for all of the possible exceptions categories on the FPC 0 and the PFE 0. The configuration is then defined to be part of the inline monitoring instance, with the name test. [1]

Code 3. Configuring the observation of exceptions at FPCs and PFEs

```
set chassis fpc 0 pfe 0 exception-reporting category forwarding-state inline-monitoring-instance test
set chassis fpc 0 pfe 0 exception-reporting category packet-errors inline-monitoring-instance test
set chassis fpc 0 pfe 0 exception-reporting category firewall inline-monitoring-instance test
set chassis fpc 0 pfe 0 exception-reporting category layer2 inline-monitoring-instance test
set chassis fpc 0 pfe 0 exception-reporting category layer3 inline-monitoring-instance test
```

For Junos OS evolved, it is not possible to select differing exception categories in the configuration, thus the configuration works by selecting the option all for exception category. Code 4 is an example, where the observation point is configured on the FPC 0 and the PFE 0, to capture all of the exceptions as part of the inline monitoring instance named test.[1]

Code 4. Observation point configuration on Junos OS EVO

```
set chassis fpc 0 pfe 0 exception-reporting category all inline-monitoring-instance test
```

2.4.3 Metering process and the Template configuration

The metering process and the customization of the information captured from the packet drop instances, is configured as part of the template configuration. The template works as an IPFIX template, using the custom information elements provided by the protocol. The example in Code 5 describes all of the possible custom information elements, which can be used to capture information from the dropped packets. These include varying interface indexes, which point to interfaces on the device, where the packet was captured, or is headed to, flow direction, forwarding class drop priority related to Quality of Services and the actual exception reason.[1]

Code 5. Template configuration

```
set services inline-monitoring template test_template primary-data-record-fields cpid-ingress-interface-index
set services inline-monitoring template test_template primary-data-record-fields cpid-egress-interface-index
set services inline-monitoring template test_template primary-data-record-fields cpid-forwarding-class-drop-priority
set services inline-monitoring template test_template primary-data-record-fields cpid-forwarding-nexthop-id
set services inline-monitoring template test_template primary-data-record-fields cpid-forwarding-exception-code
set services inline-monitoring template test_template primary-data-record-fields ingress-interface-snmp-id
set services inline-monitoring template test_template primary-data-record-fields egress-interface-snmp-id
set services inline-monitoring template test_template primary-data-record-fields direction
```

In addition to the information elements the template has to also be configured with refresh rate, which defines the update time of when the template is sent to the collector and the template id, which can be used to differentiate between differing templates. The Code 6 demonstrates how aforementioned components are configured.[1]

Code 6. Template id and refresh rate configuration

```
set services inline-monitoring template test_template template-refresh-rate 600
set services inline-monitoring template test_template template-id 1024
```

The last configuration for the template and the metering process involves configuring the template to be used as part of the respective inline monitoring instance. The example in Code 7 defines the test_template to be used as the template for the inline monitoring instance called test.[1]

Code 7. Configuring the template as a part of the inline monitoring instance

```
set services inline-monitoring instance test template-name test_template
```

2.4.4 Juniper Resiliency Interface Exceptions and Exception codes.

For reporting different exceptions Juniper uses a custom information element called exceptionCode. This custom information element can achieve differing values, which are used to represent different exceptions.

For Junos OS and OS Evolved, the exception code values are different. As for Junos OS exceptions, the values are reported as part of the exception categories, defined at the observation point level. These exception codes are thus divided by their respective exception categories. Examples of the exception codes for the Junos OS are found on Table 2 and Table 3. Here the tables represent different exception categories and examples of their exception codes. [1]

Table 2. Examples from packet error category. [1]

Code	**Exception**	Exception explanation
2	bad ipv4 hdr checksum	IPv4 checksum verification has failed.
21	mtu exceeded	Maximum transmission unit is exceeded.

Table 3. Examples from Firewall category. [1]

Code	**Exception**	**Exception explanation**
67	firewall discard	Firewall in the packet path has a action
114	firewall discard V6 out	Firewall has a action for the outgoing packet.

The exceptions for Junos OS also include the forwarding state, layer 2 and the layer 3 exception categories. But rather than listing all the possible options, the goal for the example is to provide explanation of how the values work in conjunction with the exceptionCode information element and the Juniper resiliency Interface.

For Junos OS Evolved the exceptions are not divided into categories and are just provided as a list of exceptions. Currently Junos OS Evolved offers 251 different exceptions it can report.[1] Examples of these are in the Table 4. It describes 3 differing exception codes for the Junos OS Evolved.

Table 4. Examples of Junos OS Evo exceptions. [1]

Code	**Exception**	**Exception explanation**
41	slu.trapcode.stp_blocked	Packet on STP blocked port
131	irp.core.trapcode.policer	Packet dropped due to policer
250	sw.egnh.cfg_pfh_trap	Discard nexthop

2.4.5 Export and Collector processes

As Juniper Resiliency Interface offers the possibility of the packet drop exceptions being exported either to a local or an external collector. The configuration differs depending on which type of implementation is being used. For local collector implementations, it is required to configure the collector as well as the export process. [1]

The example in Code 8 describes how to configure the local collector. In this example, the local collector is configured with the name test_store, its size is 1 gigabyte, and it allows for two files to be produced, before the old ones are deleted.

Code 8. Local collector configuration

```
set system resiliency store fwding-file test_store
set system resiliency store fwding-file test_store size 1g
set system resiliency store fwding-file test_store files 2
```

For the export process the configuration for local- and external collector implementations are similar. The configuration works by configuring the details of the collector. Example in Code 9 describes the configuration of a local collector implementation. This configuration works as a part of the already defined inline-monitoring instance called test. Firstly, the collector is defined with a name. In this instance it is defined as the test_collector. After this, it is defined with the source- and destination IPv4 addresses, as well as port and the dscp bits. [1]

Code 9. Collector details for export process

```
set services inline-monitoring instance test collector test_collector source-address 10.0.0.1
set services inline-monitoring instance test collector test_collector destination-address 127.0.0.1
set services inline-monitoring instance test collector test_collector dscp 21
set services inline-monitoring instance test collector test_collector destination-port 4739
```

To configure the exporting process for an external collector, requires the configuration of the details to point at the external collector.

2.4.6 Firewall Filter and the Reporting Loop

Although the local collector implementation uses a local log file to store the packet drop data, the IPFIX records are sent to the store as network traffic within the device. One could get the assumption that the records are simply written onto the file, but they are not. Juniper networking devices are often protected from unwanted traffic with firewall filters, and it is important to modify these protections to allow for the IPFIX exception traffic. Not doing so could result in a reporting loop, where the exported exceptions are dropped by the firewall, which in turn would produce a new exception to be exported [16].

The reporting loop only produces a one-to-one stream of exceptions. This means that for each exceptions dropped by the firewall, only one new exception

would be produced. This production of exceptions would then be capped out at the sampling rate at which the exceptions can be produced. This of course only applies to one exception occurring on the device. In a real-world instance, the router would also experience actual exceptions, which all would produce their own reporting loop and in the worst case this could result in the router experiencing network congestion, dropping actual traffic, and eventually shutting down from over utilization.

Juniper routers have an alternative safety feature, which could protect it from the packet drop loop occurring. The routers are equipped with Denial-of-Service feature, which works by limiting the rate at which certain traffic can be accepted. This is enabled by default for all of the possible protocols and the exception traffic is included in the default setting [19]. The default configuration can however be modified, thus it is important to ensure the safety configuration and the features of the devices before implementing the JRI feature.

3 System Design

The system architecture was designed to align with the company's existing products and to use the already available tools. The available tools include LXD container hosts, the Grafana platform and the Juniper routers.

Another factor, which affected the system design, was the fact that JRI uses UDP protocol for the transportation of the IPFIX packets. UDP as a transport protocol does not ensure the delivery of the packets [20]. The protocol is widely used in video and voice applications, where the low latency and reduced overhead are more important than the reliable transportation. As for the thesis implementation, the reliable transportation is considered more important. It could also be seen as illogical to send information related to packet drops using a protocol that does not take packet drops into consideration.

These factors resulted in a system design, which works on a pull architecture. This means that the packet drop recordings are pulled from the routers by a Python program instead of the routers pushing the recordings to an external collector. This allows for the data collection to use the TCP transport protocol, which ensures the transportation of the packets. The Figure 4 Demonstrates the top-level system design. The system is composed of five major components. Routers, LXD container, Python program, PostgreSQL database and the Grafana platform. Each of the components will be introduced in this chapter.

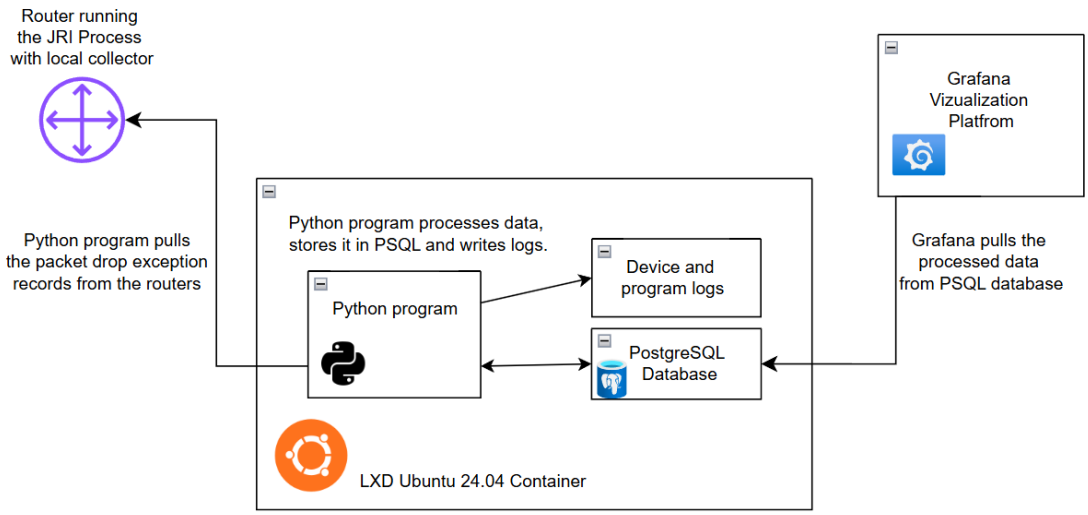


Figure 5. Illustration of the system design

3.1 Juniper routers

The first phase for the system design starts with the routers. As Juniper offers the possibility of recording the packet drop exceptions either locally or to an external collector, the local collector is chosen as the more preferable option for this implementation.

The reason for favoring the local collector implementation is that it offers higher integrity for the data. As the external collector implementation would send the data over the network using UDP protocol, during the time when the router or the network is experiencing packet loss. There could be a possibility of the packet drop data itself being dropped. The local collector implementation also uses UDP to send the data to itself and there is a possibility of it being dropped too. The packet, however, has to only travel within the device itself, resulting in a greater survival rate.

For the thesis work, the code was tested on varying device models in a network laboratory environment. As the configuration for the Junos OS and Junos OS Evolved differs, and there are both device types in network. Configuration had to be prepared for both device types. Both of the configurations offer the same

level of service, differing only in the exception categories and the exceptions produced.

3.1.1 Junos OS Configuration

For the thesis implementation the Junos OS configuration follows the default options offered by Juniper. The exception category configuration is applied for all of the categories on all of the FPCs and PFEs on the device. It only uses one inline monitoring instance and one template. The collector is configured as a local collector, its size is 50MB and it produces two store files, before the old ones are deleted. In this manner the configuration is simple and can be applied to all of the Junos OS routers globally.

3.1.2 Junos OS Evolved Configuration

The configuration for the devices running the Junos OS Evolved the configuraiton also follows the default options provided by Juniper. Only difference compared to the configuration of the devices running the Junos OS is the configuration of the exception category. For Junos OS Evolved devices the exception category option all is selected.

3.2 Python

Python is a high-level object-oriented programming language offering users extensive selection of libraries and great readability. It is one of the most used programming languages in the world.[22]

Python is used in the thesis implementation as the main programming language. In the JRI implementation Python is used for extracting the packet drop exceptions from the routers, processing the data to a more usable format, and saving the data into log files and into a PostgreSQL database.

The reason for deciding to use Python was due to its large selection of libraries. Many of the features in the program are explicitly implemented from existing python libraries.

Although Python as an interpreted programming language is not regarded as a fast-programming language, it proved to be fast enough for this implementation. The greatest time consumer in the program is the SCP function, which is used to collect the logs from the devices and the operation time for it is not related to the program, but rather network and the devices in it.

3.2.1 Python Libraries

The thesis program makes use of Python libraries to execute various tasks. Examples of the tasks include time-related tasks, database connection, operating system functions and the SSH connection. The program uses the following Python libraries:

- Datetime
- Logging
- Os
- Paramiko
- Psycopg2
- Python-dotenv
- Pytz
- Socket
- Time

3.2.2 Python Program

The focus for the program was to build a self-sufficient program, which can be easily imported to a new platform if needed. As the program will be conducting tasks throughout the day, it should not require any user input.

This chapter will explain the functionalities of the built program. It will explain it in the order the program is executed.

Program variables

For the program to work there has to be some prerequisite variables set. These include user credentials, database information and in this instance, the query commands which are used for acquiring the list of routers running the JRI service.

The variables are set in a .env environment variables file. Variables from this file are imported to the program using the python-dotenv library features. Using the .env file for variables makes both the program easier to manage and safer, as the sensitive information does not have to be written directly into the code [23].

Other variables, which are set in the program itself are the date related variables used for logging related purposes. These are used in naming the log files during the creation and later for opening the log files.

Function program_logging

This function is used for writing logs about the program execution events. This includes information such as execution time, errors, device information and more. The function uses the Python logging library to provide the logging framework and the input calls to the function are called from the other functions of the program.

The function produces log files on a daily basis. The name for the log files is acquired from the current_date variable set earlier in the program, thus the program will produce a new file every day. The Figure 5 demonstrates the program log file structure constructed by the program.

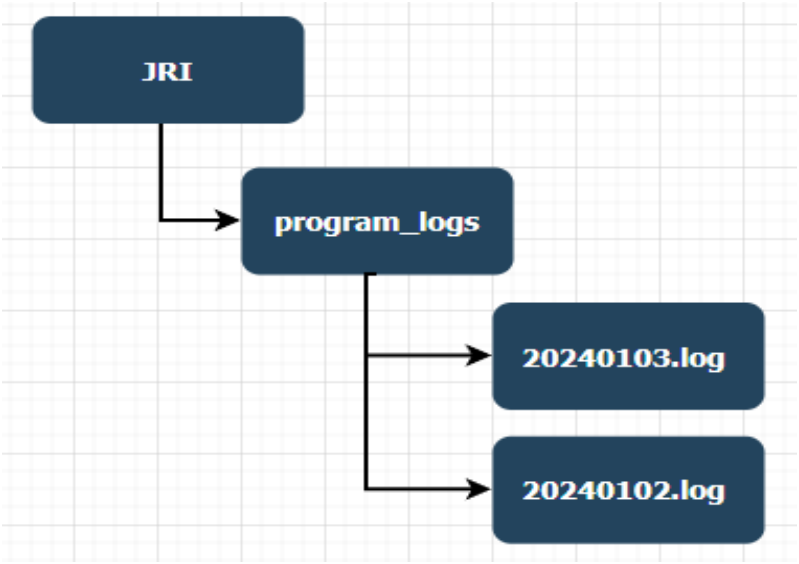


Figure 6. Illustration of the program log structure.

The program logs can be used for debugging the program behavior. The Figure 7 and the Figure 8 illustrate the program logs produced from successful and non-successful program executions.

```
2025-03-25 12:18:58,028 - INFO - Started collecting logs from 10.0.0.1
2025-03-25 12:18:58,028 - INFO - Connecting to host 10.0.0.1
2025-03-25 12:18:58,669 - INFO - Attempting to collect the logs with SCP
2025-03-25 12:18:58,925 - INFO - Collected the logfile on device_1
2025-03-25 12:18:58,949 - INFO - It seems that device_1 has not been collecting packetdrop data before.
2025-03-25 12:18:58,950 - INFO - Started parsing the log files from device_1
2025-03-25 12:18:58,981 - INFO - Creating a database table for device_1
2025-03-25 12:18:59,054 - INFO - Inserting parsed data from device_1 in to the database
2025-03-25 12:18:59,081 - INFO - Program run took 1.0526399612426758 for the device_1
```

Figure 7. Example of the program log file, from a successful execution.

```
2025-03-25 12:19:52,187 - ERROR - Error occured in program for device device_2: scp: /var/log/test_store.log: No such file or directory
2025-03-25 12:19:52,187 - INFO - Started collecting logs from 10.0.0.3
2025-03-25 12:19:52,187 - INFO - Connecting to host 10.0.0.3
2025-03-25 12:19:52,759 - INFO - Attempting to collect the logs with SCP
2025-03-25 12:19:52,930 - INFO - Collected the logfile on device_3
2025-03-25 12:19:52,950 - INFO - It seems that device_3 has not been collecting packetdrop data before.
2025-03-25 12:19:52,950 - INFO - Started parsing the log files from device_3
2025-03-25 12:19:52,950 - ERROR - Error occured in program for device device_3: No exceptions in the test_store on the device_3
```

Figure 8. Figure demonstrates an example from the program log file, where an error occurred in the program execution.

Function device_logging

The function is used for writing log files from the packet drop exceptions acquired from the devices. The function also uses the Python logging library for the logging framework. Input calls for the function are called using the parsed

packet drop data acquired form later functions. The Figure 9 illustrates the contents of the device log file. In the contents of the file there are entries of two different types of exceptions that occurred on 25th of March.

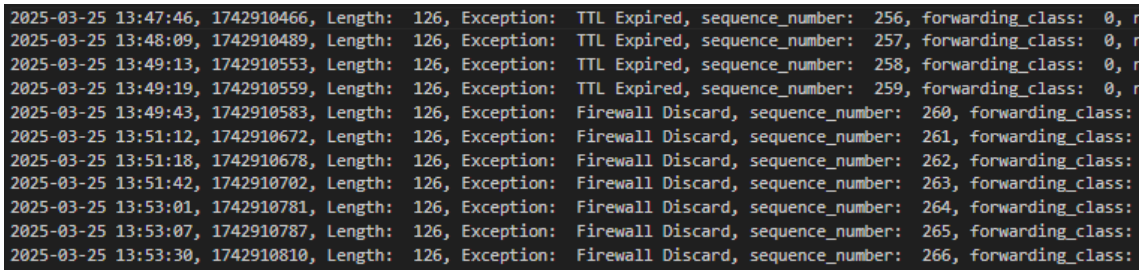


Figure 9. Example from a device log file.

The function uses the current_month variable and the name of the device to build the device log structure. As the log file uses current_month as a file name, new files are produced on monthly basis. Figure 6 demonstrates how the Python program builds the file structure for each of the devices.

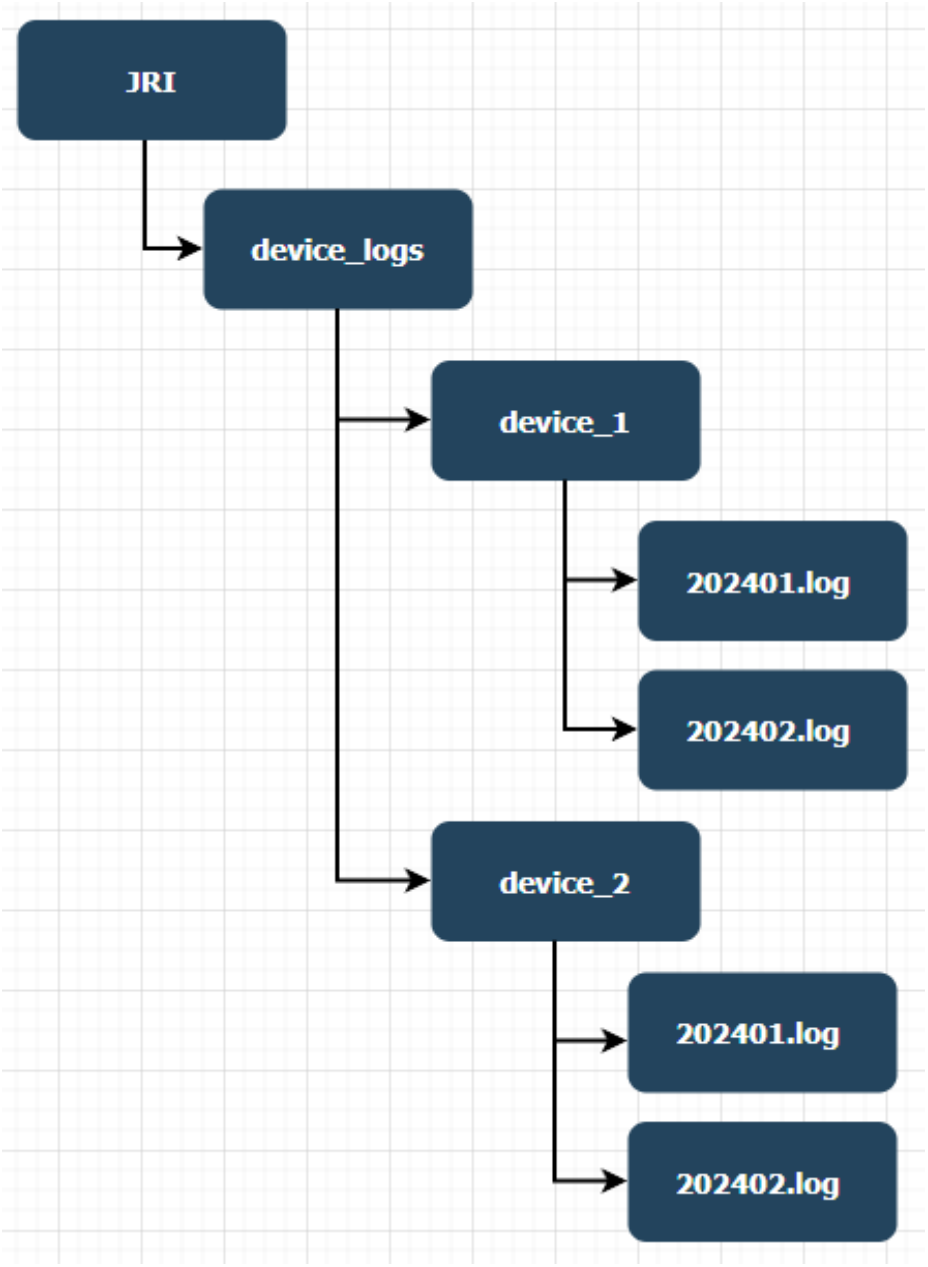


Figure 10. Device logging structure.

Function get_list_of_devices

The get_list_of_devices function is used to gather the updated list of routers that are running the Juniper Resiliency Interface feature. As the devices that are running the service are subject to fluctuation, it is more logical to gather the list dynamically than to use a static list.

The function works by querying the information from the device database and building a Python list out of the answer.

Function collect_logs_from_devices

The collect_logs_from_devices function is used for collecting the packet drop logs from the routers. It uses the SCP file copy feature provided by the Paramiko SSH library[24]. The function takes as an input the IPv4 address of a device, logs into it and copies the packet drop log file from the router to the current working directory.

The second feature of the function is gathering the hostname of the device for further use. As the devices in DNAs network have unique hostnames, it is cleaner to use the hostnames in other features of the program, such as logging, log file naming and database functions. The hostname is gathered while the function is connected to the device, by passing a command to the router.

Function get_last_timestamp

The function is used to gather the timestamp of the latest packet drop exception entry from the database table. It essentially also tests the connection to the PostgreSQL database and ensures the existence of the table.

As the program gathers the packet drop exception log file from a router, it is expectable that the log file will contain same exception that might have been already acknowledged during the earlier executions of this program. The last timestamp is thus used in the program to ensure that it does not parse any exception entries that have already been acknowledged and inserted into the database.

The last timestamp is also used in the program to inform if the log files on the routers are filling up too quickly. If the last timestamp gathered from the database does not exist in the new log file collected from the router, it proves that there are too many exceptions occurring on the device and program is not able to gather all the exceptions. If this occurs, it is informed in the program log file.

Function parse_data

The parse_data function is used to parse the acquired data into a more usable format. It does this by reading the log file gathered from the device, identifying all the new exceptions, their information elements and building a list of dictionaries from them.

The collection of information element data pairs is done in this function by directly addressing the indexes in the exception packet, rather than just looping over the exception packet and gathering it. The Figure 7 demonstrates how the specific elements in the packet are addressed in the code. This facilitates the possibility of the code being easier to modify in the future, if the IPFIX packet should change.

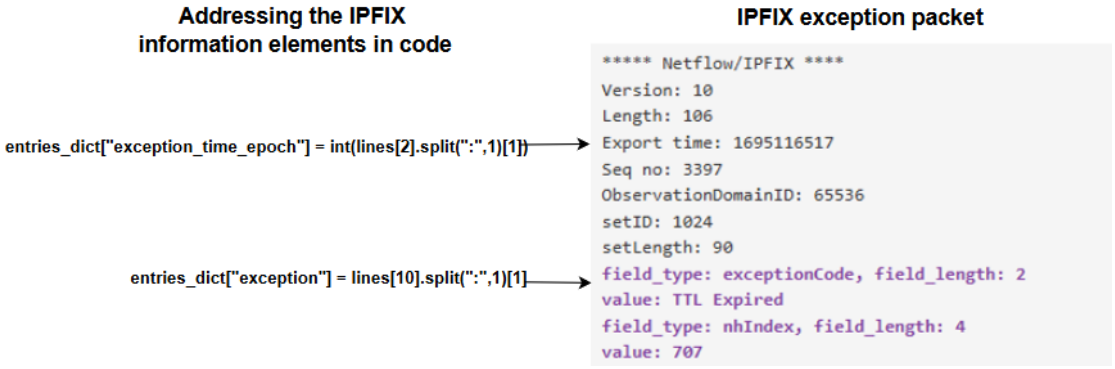


Figure 11. The information elements being addressed in the code.

The IETF draft co-authored by Juniper Engineers requesting new information elements to be added to the IPFIX protocol, implies that the Juniper Resiliency Interface exception packet and its format might also be subjectable to change [14]. This was acknowledged while writing the program, and this partly led to it being programmed in that manner.

Function write_database

The write_database function is responsible for pushing the new data into the database. This function uses the Psycopg2 Python library as the adapter for connecting to the database.

The function takes as an input the parsed data and the variable defining the existence of the database table. In an instance, where the program has just collected exception logs from a device for the first time, the function will also make a database table for this device.

3.2.3 Exception Handling

As the program works with many components, which all have a possibility of experiencing faults, there has to be some exception handling mechanics involved. The functions within the program are mostly written with try and except clauses, allowing for defining the exception mechanics [25].The Figure 8 demonstrates the how the exception handling is conducted in varying parts of the program.

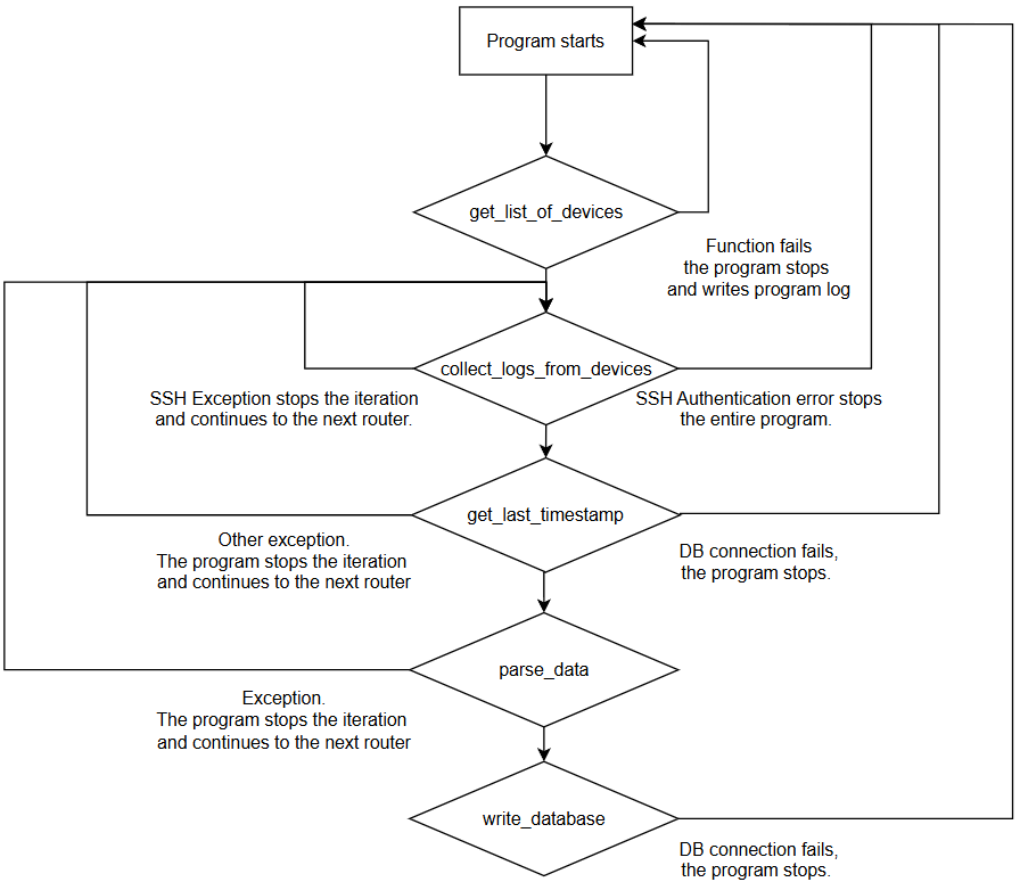


Figure 12. Illustration of the exception handling mechanics of the program.

Essentially the exceptions related to the program and its components can be divided into two groups. Exceptions, which demand for the whole program to stop and exceptions, which are only related to device instance.

There are three exceptions where the whole program stops. Firstly, the fault in gathering the list of devices automatically stops the program. In this instance there would be nothing for the program to work on and it stops. The second exception is SSH authentication exception. The program has its own user credentials, which are used for authentication in all of the SSH functions. If there is an error related to it, it can be assumed that there is something wrong with the user and the program should be stopped. The third exception leading to stoppage of the program is related to database connectivity. As both the database and the program are running on the same host, connectivity issue related to database can be considered as a major fault. For this it can be considered more logical to stop the program than to allow it to keep attempting connection for each router iteration.

After acquiring the list of hosts running the JRI process, the program starts processing the packet drop exceptions a device at a time. This involves connecting to the device, gathering logs, writing data and more. The second category of exceptions are related to these functions. For these functions it is not reasonable to stop the entire program, but rather just allow the program to proceed onto another device from the device list.

These exceptions include file management errors, OS errors, SSH exceptions related to routers and new log files not containing exceptions. A good example of an exception could be a router, which is under maintenance and cannot be reached over the network. In this instance the program would just skip the router under maintenance and continue on to the next router from the list.

3.3 LXD containers

LXD is a Linux based containerization implementation. It has some key differences compared to Docker containers, which are often meant when talking about containers. LXD containers are efficient and lightweight but also offer the ability to have a full operating system, working more like a virtual machine. [26]

LXD is used in the thesis implementation, for hosting the database and the Python program. The reason for using LXD containers, instead of other technologies, such as docker or virtual machines is that it offers a full operating system, while still being lightweight. This serves a good base to run both the Python program and the database on.

3.3.1 LXD configuration

As the container environment is already built, the only configuration related to the LXD containers involves the specific container configuration. The base for the LXD container implementation is the software image. For this implementation, the container was built using the Ubuntu 24.04 operating system. The reason for using Ubuntu is the popularity and the personal familiarity of the operating system.

The configuration of the container follows a similar configuration, as with other LXD containers used in DNA and this container environment. This entails configuring the container with its own network and storage. This assists in making clear separation between the other live containers running in the system and configuring the network will be safer, since changes will not affect the other container instances.

The storage configuration includes creating a storage and attaching it to a container. Having separate storage from other containers enforces a better isolation and allows for easier storage modification in the future.

3.3.2 LXD Network configuration

LXD offers a wide selection of possibilities for configuring the containers networking. Most notable options are the top-level networking technologies, such as Bridge, OVN, Macvlan, SR-IOV and a physical network. Most common types used are Bridge and Macvlan. [27]

For the thesis program Bridge network is used. It works like a Layer2 Ethernet Switch offering the possibility of connecting multiple containers to it. The bridge network is equipped with DNS process, which provides DHCP, IPv6 route announcement and DNS service for the containers using the bridge network [28]. The benefit of using a bridge network is the clear separation from other networks and other containers. This facilitates a safer environment for maintenance. Another benefit is that in the future, if other containers are introduced to the system, it is easy to just attach them into the existing bridge network. An example of this could be a container which offers redundancy for the service JRI service. The example in the Code 12 demonstrates the LXD configuration of the thesis environment.

Code 10. LXD Network Configuration

```
lxc network create <network name>
lxc network edit <network name>
config:
   ipv4.address: 192.168.1.100/24
   ipv4.nat: "true"
   ipv4.routes: 10.10.10.10/32   <- route to the static IP address
   ipv6.address: none
   ipv6.nat: "false”
 description: ""
 name: <network name>
 type: bridge
 used_by:
 - /1.0/instances/<container name>
 managed: true
 status: Created
 locations:
```

One of the most notable network configuration element is the static public IPv4 address. As one of the main operations for the program is connecting to the routers for collecting the logs, the program has to have a static public IPv4

address. This static address is then configured on the firewalls to allow for connection to the routers. The public IP address is configured on the container as a secondary address and the LXD container network is configured with a route to the address. This public IP address is configured as a secondary IP address for the container and the route to it is configured as part of the LXD network configuration.

For other connections outside of the container, it will use its IP address acquired from the DHCP process.

3.4 PostgreSQL & TimescaleDB

PostgreSQL is an open-source object-relational database developed by the computer science department of the University of Berkley. It works in accordance with the SQL standard [29]. SQL or Structured Query Language is a standard and a programming language used for managing relational databases [30].

Data in PostgreSQL databases is structured in tables. Tables consist of rows and columns, which are then populated with data. PSQL offers a large selection of data types, for allowing us to store data in its most logical datatype. Examples of data types include integers, big integers, dates, strings and more. [31]

Timescale DB is an open-source time-series database working as an extension for PostgreSQL. Compared to traditional PostgreSQL, it offers faster performance in time related functions. Its most notable feature is the hypertables, which provide automatic data partitioning, by splitting the database tables into smaller time-related logical batches. This allows for faster querying for information related to certain time windows. [32]

For the thesis implementation PostgreSQL and Timescale DB is used as the database to store the packet drop exception data. The reason for deciding to use This for the implementation was due to its open-source status, the Python

Psycopg2 library and the fast performance of the Timescale DB. Together they provide a well manageable and efficient combination for data related to the thesis work.

3.4.1 Database Configuration and Structure

The configuration for the PSQL database follows a minimal setup for Ubuntu. Steps for the minimal setup include installing it, starting the process, and switching the default postgres user's password. The Timescale DB is then installed as an extension on top of the PostgreSQL database.

The configuration of the database itself is done by the Python program. It builds the database table, the timescale hypertable and the data columns for the database. This allows for the program to be installed into a new system more easily.

The data is stored in the database on one table and eleven columns as described in the Table 5. The columns include all of the information captured from the packet drop occurrences, such as exportation time, exception code and the interfaces involved. Data is stored in the database in four different datatypes. Focus was to use the most memory-efficient data types possible. For the exception_time_epoch and the sequence_number the BIGINT datatype was used. For these the datatype INT would not have been enough, since they have the possibility of acquiring larger values.

Table 5. Example of data types and the columns used in database

Column	**Data type**
exception_time	TIMESTAMPTZ
exception_time_epoch	BIGINT
exception	VARCHAR
observation_domain_id	INT
sequence_number	BIGINT

Column	Data type
forwarding_class	INT
nexthop_index	INT
oif_index	INT
ingress_interface_index	INT
ingress_interface	INT
egress_interface	INT

3.4.2 Database Users and Access

The database access is conducted in a least privilege manner. This entails that the users and hosts are given only the minimal required use privileges to the database. For this implementation two database users are made.

The first database user is used by the Python program. This user's task is to create database tables, query information and insert information to the database. For this reason, the user was given permission to create, insert and to query the data. As both the program and the database reside on the same host. The user does not have access to log into the database, from outside of the host.

Second user for the database is used by the Grafana platform. Its task is to query information from the Database and to visualize it. As the user will only query the information, its privileges are limited to SQL SELECT functions. This means that the user cannot modify the data. Access to the database is also only permitted for the user, from the IPv4 address of the Grafana server. This means that the user could not be used to access the database from other IP addresses.

3.5 Grafana

Grafana is an open-source platform offering tools for visualizing data. It allows for querying data, building dashboards, creating alerts and more[33]. The data

can be imported into Grafana from sources, such as Prometheus, PostgreSQL, Influx DB, Elasticsearch and others. [34]

In the thesis implementation Grafana is used for visualizing the data from the packet drop exceptions. One of the most valuable information that can be acquired from the packet drops is the time-related trends and the deviations from the network baseline. For visualizing this, Grafana offers Time series tools. Time series graphs are used to visualize variations in the data values over a certain time [35].

3.5.1 Grafana Configuration

As DNA already hosts a local implementation of the Grafana platform. The configuration for Grafana as part of the thesis only includes the configuration of the data source. The data source is the PostgreSQL database built by the program. Grafana also allows for the use of the enhanced time-related functions provided by the Timescale DB extension. This can be configured as part of the PostgreSQL data source. The Figure 13 is an example of configuring a PostgreSQL as a data source and enabling the TimescaleDB as an extension for it.

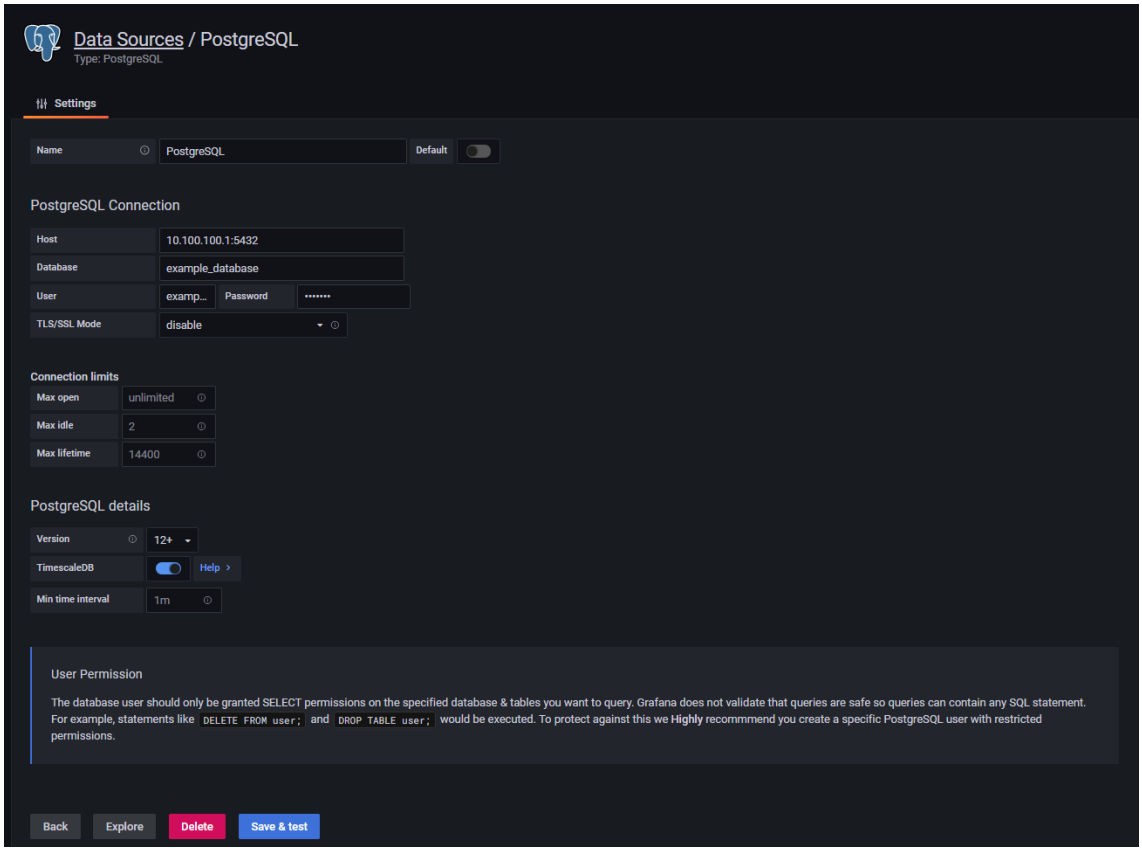


Figure 13. Illustration of configuring the PSQL data source.

3.5.2 Grafana Dashboards

The packet drop data is visualized in Grafana with two panels. The two panels offer both the broader trend lines of the packet drop occurrences as well as the specific data in a table view. This allows administrators to notice deviations in network baseline and to see exactly what packet drops are occurring, within the same dashboard.

Configuring the panels can be achieved by either using the query builder provided by the Grafana, or by applying direct database queries. For the thesis panels, SQL queries were used. For PostgreSQL data sources Grafana also provides macros, which can be used for time related functions.[34] For the thesis $__TimeGroup macro was used for inserting the time windows used in the dashboard into the database queries. This allows for relatively faster queries if a shorter time window is selected for the dashboard.

The first panel visualizes the amount of combined packet drops occurring on a device in a 30-minute window. This is visualized as a time series panel, where a line is drawn for each router. It does not illustrate the separate packet drop categories but rather counts all of the occurrences. The reason for this approach is that the panel would get crowded if it had to draw a line for each of the packet drop types, for each of the devices.

Configuring the time series panel is conducted with the following SQL query in the Code 13. It queries the count of occurrences in the exception_code panel, where the hostname is the same. It counts the occurrences in 30-minute windows and draws a line for each of the differing devices. All of this is ordered by the exception_time panel.

Code 11. SQL Query for the Time Series panel

```
SELECT $__timeGroupAlias(exception_time, 1m), hostname, count(exception_code) AS " "
FROM exceptions
WHERE $__timeFilter(exception_time)
GROUP BY 1, hostnames
ORDER BY 1;
```

The second panel is a table, which can be filtered by attributes and time. It queries all of the information from the database using the time window provided by the $__TimeGroup macro. The purpose of this panel is to allow for filtering of detailed information from packet drop occurrences.

The configuration of the table view is conducted with a SQL query described in Code 14. It returns all of the database entries from certain time window and by enabling the filtering of columns from Grafana table configuration.

Code 12. SQL Query for the table panel

```
SELECT *
FROM exceptions
WHERE $__timeFilter(exception_time)
ORDER BY exception_time;
```

3.6 Security Measures

The direct safety features, related to DNA:s organizational implementations cannot be published, but these include using firewalls, strong authentication, network segmentation, among others. For the thesis implementation some safety features were extended, such as using the least privilege principle for functions and users.

4 Key Results

This chapter will present the key results achieved from testing the feature. These results include the actual data results, effects on the routers and program performance results.

The testing of the system was conducted in a lab environment, where the feature was implemented on multiple Juniper routers. The routers included varying models, running both the Junos OS and the Junos OS Evolved. For some of the routers the JRI storage was manually modified, to allow for testing of the program exception handling mechanics.

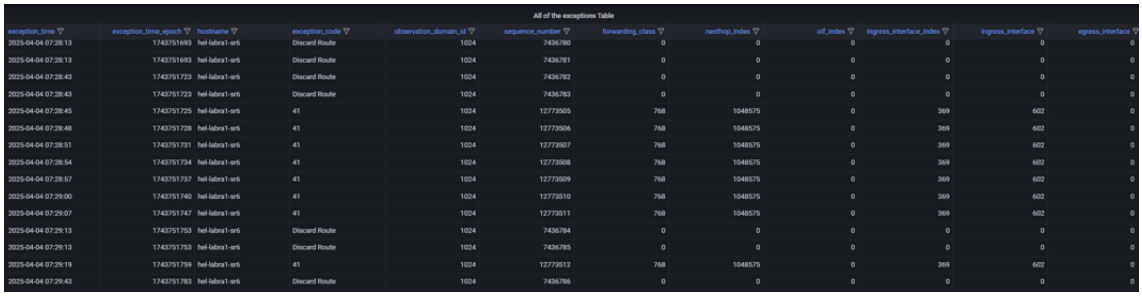
The actual exception data for the feature was produced both by manually producing traffic that required the packets to be dropped, and by the actual network traffic in the lab environment. Although the traffic in the lab environment is small, it still produced some baseline packet drop data and occasional packet drop spikes.

4.1 Data results

Most notable result achieved from the system is the visualized data in the Grafana platform. The dashboard is built in a manner, where clear deviations in packet drops can be located in the time series panel and then the actual data can be studied from the table view below. The example in the Figure 14 illustrates a packet drop trendlines being drawn of exceptions from four different routers. A clear deviation can be seen on the blue trendline describing packet drops on certain device.

Figure 14. Example of the packet drop trends.

The Figure 15 illustrates a table view of the packet drop data. In the Grafana dashboard the table view is located directly below the timeseries panel, allowing for examination of specific packet drops.


All of the exceptions Table

exception_time	exception_time_epoch	hostname	exception_code	observation_domain_id	sequence_number	forwarding_class	nexthop_index	oif_index	ingress_interface_index	ingress_interface	egress_interface
2025-04-04 07:28:13	1743751693	hel-labra1-sr6	Discard Route	1024	7436780	0	0	0	0	0	0
2025-04-04 07:28:13	1743751693	hel-labra1-sr6	Discard Route	1024	7436781	0	0	0	0	0	0
2025-04-04 07:28:43	1743751723	hel-labra1-sr6	Discard Route	1024	7436782	0	0	0	0	0	0
2025-04-04 07:28:43	1743751723	hel-labra1-sr6	Discard Route	1024	7436783	0	0	0	0	0	0
2025-04-04 07:28:45	1743751725	hel-labra1-sr6	41	1024	12773505	768	1048575	0	369	602	0
2025-04-04 07:28:48	1743751728	hel-labra1-sr6	41	1024	12773506	768	1048575	0	369	602	0
2025-04-04 07:28:51	1743751731	hel-labra1-sr6	41	1024	12773507	768	1048575	0	369	602	0
2025-04-04 07:28:54	1743751734	hel-labra1-sr6	41	1024	12773508	768	1048575	0	369	602	0
2025-04-04 07:28:57	1743751737	hel-labra1-sr6	41	1024	12773509	768	1048575	0	369	602	0
2025-04-04 07:29:00	1743751740	hel-labra1-sr6	41	1024	12773510	768	1048575	0	369	602	0
2025-04-04 07:29:07	1743751747	hel-labra1-sr6	41	1024	12773511	768	1048575	0	369	602	0
2025-04-04 07:29:13	1743751753	hel-labra1-sr6	Discard Route	1024	7436784	0	0	0	0	0	0
2025-04-04 07:29:13	1743751753	hel-labra1-sr6	Discard Route	1024	7436785	0	0	0	0	0	0
2025-04-04 07:29:19	1743751759	hel-labra1-sr6	41	1024	12773512	768	1048575	0	369	602	0
2025-04-04 07:29:43	1743751783	hel-labra1-sr6	Discard Route	1024	7436786	0	0	0	0	0	0

Figure 15. Grafana table representing all of the packet drop exceptions.

While the data produces meaningful information about the packet drops, it does not describe the whole picture. There is some caveats, such as not knowing the underlying protocols, source- and destination IP addresses. This results in a situation where we know why, when and where the packets are dropped, but not exactly what kind of traffic is being dropped. As IPFIX protocol is equipped with the information elements that could provide such information, it can only be wished that the JRI feature is extended to provide this information in the future.

Another caveat is related to the exception types produced by the routers. For Junos OS there are only twenty-three different exception types available and for Junos OS Evolved, there are approximately two hundred different exception types available. They offer visibility into the exceptions but cannot be considered to have the ability to record all of the packet drops. An example of this could be a software-related bug, where the packets are dropped due to an unknown fault and the JRI does not have the ability to record these packet drops.

4.2 Effect on routers

As new features are implemented on the routers, it is important to verify how the new features affects the device. During the thesis work the load produced by the Juniper Resiliency Interface was verified from the system load charts

produced by the routers. The load produced by the JRI feature was verified both during the time, when feature was first implemented on the router and when the router experienced greater amounts of packet drops.

Surprisingly, the feature had a minimal effect on the routers. This was verified from the system load charts from multiple routers and almost all of them had only a minimal deviation from the baseline.

The Figure 16 illustrates a view of a Grafana time series table from when the feature was first implemented on the routers. During the following four hours after the initial implementation the device experienced over a hundred thousand exceptions. This is represented by the orange trendline, which clearly deviates from the baseline.

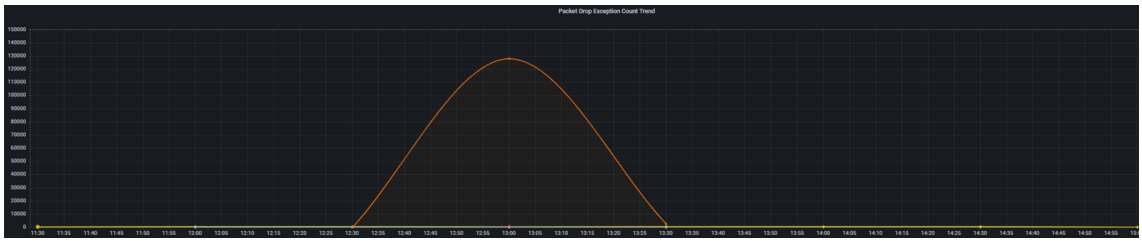


Figure 16. Grafana timeseries table during the feature implementation.

The Figure 17 illustrates the routers CPU utilization during the same timeframe, from when the feature was initially implemented and when the hundred thousand errors occurred. It can clearly be seen that there is no deviation from the baseline during this time.

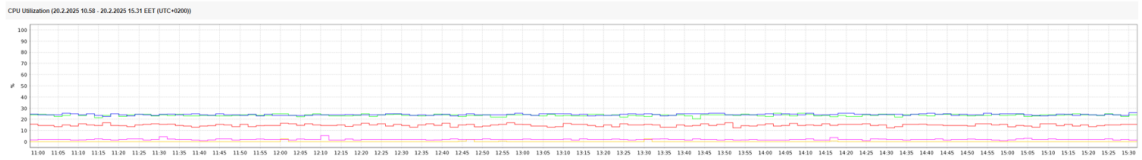


Figure 17. Router CPU utilization during the feature implementation and large amount of packet drops.

The Figure 18 illustrates the buffer memory utilization from the aforementioned timeframe. There is no deviation from the baseline.

Figure 18. Router buffer memory utilization during the feature implementation and large amount of packet drops.

As this was tested in the laboratory environment where the network traffic is small and the devices are not under extreme loads, there is a possibility of results varying in actual network implementations.

4.3 Program performance

During the laboratory testing the program was evaluated for the program efficiency, the exception handling mechanics and for the integrity of its data related functions. This was conducted by manually producing differing data and program errors.

The program efficiency was evaluated in terms of its execution time. This was conducted by producing different sized packet drop logs on the routers and then measuring the execution time of the program. The tests proved that the program could process approximately 100MB of packet drop data in one minute. The largest time consumer for the program is the SCP function, where the packet drop data is transported to the program container and the actual parsing of the data.

The program exception handling mechanics were assessed for the most probable errors. These included connection errors, file errors, database errors and other data related errors. Since the program was built to manage these, they did not affect the program in an undesired way. This allows for self-sufficient execution.

The integrity of the data was verified by comparing the raw packet drop data from the routers to the one parsed by the program. More precisely it was conducted by selecting exceptions from the packet drop logs on the routers and

ensuring that the exact exceptions were also found in the database, and the device logs produced by the program. This proved that the Python program produces accurate data. The only instance where the packet drop data could be lost is either if the router itself could not record the packet drops or when there are too many packet drops occurring on the router and the Python program cannot keep up. If the latter occurs, the program will however inform the user about this in the program logs.

5 Conclusions and Future Development

The first goal of the thesis was to familiarize with the Juniper Resiliency Interface and although as a new technology it is not yet largely documented, the technologies and protocols it is based on provided the missing information. The theoretical study was initially focused on the feature itself and later on technologies, such as flow monitoring and IPFIX. Although the Juniper Resiliency Interface is not directly a flow monitoring technology, it relies on these technologies and understanding them helps in understanding the Juniper Resiliency Interface.

The second part of the thesis focused on developing a system which records, collects, parses, saves, and visualizes the packet drops. This objective can also be considered as achieved, although the work on this system will continue. This part of the work was conducted in the laboratory environment, where it proved to work self-sufficiently and managed most of the exceptions introduced.

In the future the system will be introduced into a live network, where the router count will rise incrementally. This will be a longer process, and it is expected that some modifications will have to be made to the system then. This was considered in the design, and it was built in a modular fashion where components can be modified without the need for the whole product to be modified. As the live network routers are under more utilization, the implementation of the feature to the production network will also require reassessing of how the feature affects the routers system load and security.

The future development will also focus on exploring other use cases for the packet drop data. In its current format, the system produces accurate packet drop data, but the usage of the data is limited only to a few visualizations panels. The packet drop data usage could be extended to more accurate visualizations panels and for producing alerts. The Grafana platform facilitates the production of alerts, and this feature could most likely be used for it.

Immediate future work will be conducted on the Python program, which works at the center of the system. It will be modified to work concurrently, which can greatly improve the execution time. Another future improvements will be related to the storage and the data storing. As more routers will be introduced to the system, the data amount is expected to rise greatly. During this incrementation time, the data amount should be followed and modifications to manage it should be made. This is achieved by incrementing the LXD storage size and modifying the time the database holds the data. As timescale DB allows for setting a time window it stores the data, this should be modified related to the needs.

References

[1] Juniper Networks, ‘Juniper Resiliency Interface | Junos OS | Juniper Networks’. Accessed: Feb. 07, 2025. [Online]. Available: https://www.juniper.net/documentation/us/en/software/junos/flow-monitoring/topics/topic-map/resiliency-exception-reporting.html

[2] S. Amancha, R. C. A. Naidu, V. R. Bolla, and K. Meghana, ‘Modern approach of detecting packet loss and recovery in the networks’, *International Conference on Electrical, Electronics, and Optimization Techniques, ICEEOT 2016*, pp. 1722–1727, Nov. 2016, doi: 10.1109/ICEEOT.2016.7754980.

[3] L. Tan, W. Su, W. Zhang, H. Shi, J. Miao, and P. Manzanares-Lopez, ‘A Packet Loss Monitoring System for In-Band Network Telemetry: Detection, Localization, Diagnosis and Recovery’, *IEEE Transactions on Network and Service Management*, vol. 18, no. 4, pp. 4151–4168, Dec. 2021, doi: 10.1109/TNSM.2021.3125012.

[4] M. S. Borella, ‘Measurement and interpretation of internet packet loss’, *Journal of Communications and Networks*, vol. 2, no. 2, pp. 93–102, 2000, doi: 10.1109/JCN.2000.6596738.

[5] J. Lucek, ‘Detection of Blackholes Using Juniper Resiliency Interface’. Accessed: Feb. 26, 2025. [Online]. Available: https://community.juniper.net/blogs/julian-lucek/2024/01/02/detection-of-blackholes-in-networks-using-jri

[6] J. Novak, ‘RFC 6645 - IP Flow Information Accounting and Export Benchmarking Methodology’, Jul. 2012, *RFC Editor*. Accessed: Mar. 13, 2025. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6645

[7] R. Hofstede *et al.*, ‘Flow monitoring explained: From packet capture to data analysis with NetFlow and IPFIX’, *IEEE Communications Surveys*

and Tutorials, vol. 16, no. 4, pp. 2037–2064, Apr. 2014, doi: 10.1109/COMST.2014.2321898.

[8] P. Aitken, B. Claise, and B. Trammell, 'RFC 7011 - Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information', Sep. 2013, *RFC Editor*. Accessed: Mar. 09, 2025. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7011

[9] S. Kashima and A. Kobayashi, 'IP multicast traffic monitoring system with IPFIX/PSAMP', in *8th Asia-Pacific Symposium on Information and Telecommunication Technologies*, Kuching, 2010, pp. 1–6. Accessed: Mar. 18, 2025. [Online]. Available: https://ieeexplore-ieee-org.ezproxy.turkuamk.fi/document/5532073

[10] Juniper Networks, 'Configuring Observation Domain ID and Source ID for Version 9 and IPFIX Flows | Junos OS | Juniper Networks'. Accessed: Mar. 18, 2025. [Online]. Available: https://www.juniper.net/documentation/us/en/software/junos/flow-monitoring/topics/task/flow-aggregation-observation-domain-id-source-id-configuring-version9-ipfix.html

[11] B. Claise and B. Trammell, 'RFC 7012 - Information Model for IP Flow Information Export (IPFIX)', Sep. 2013, *RFC Editor*. Accessed: Mar. 18, 2025. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7012#section-2.1

[12] Juniper Networks, 'Configuring Inline Active Flow Monitoring to Use IPFIX Flow Templates on MX, vMX and T Series Routers, EX Series Switches, NFX Series Devices, and SRX Series Firewalls | Junos OS | Juniper Networks'. Accessed: Mar. 19, 2025. [Online]. Available: https://www.juniper.net/documentation/us/en/software/junos/flow-monitoring/topics/concept/services-ipfix-flow-template-flow-aggregation-configuring.html

[13] IANA, ‘IP Flow Information Export (IPFIX) Entities’. Accessed: Mar. 19, 2025. [Online]. Available: https://www.iana.org/assignments/ipfix/ipfix.xhtml

[14] C. Munukutla, S. Vaid, A. Mahale, and D. Patel, ‘IP Flow Information Export (IPFIX) Information Elements Extension for Forwarding Exceptions’, Sep. 19, 2019, *IETF*. Accessed: Feb. 26, 2025. [Online]. Available: https://www.ietf.org/archive/id/draft-mvmd-opsawg-ipfix-fwd-exceptions-08.html

[15] A. Elita, ‘Packets Lost in Transit?’ Accessed: Feb. 14, 2025. [Online]. Available: https://community.juniper.net/blogs/anton-elita/2024/01/08/packets-lost-in-transit

[16] Juniper Networks, ‘Inline Monitoring Services Configuration | Junos OS | Juniper Networks’. Accessed: Mar. 10, 2025. [Online]. Available: https://www.juniper.net/documentation/us/en/software/junos/flow-monitoring/topics/topic-map/inline-monitoring-services-configuration.html

[17] Juniper Networks, ‘Top Differences Between Junos OS Evolved and Junos OS | Junos OS Evolved | Juniper Networks’. Accessed: Feb. 26, 2025. [Online]. Available: https://www.juniper.net/documentation/us/en/software/junos/overview-evo/topics/concept/evo-top-differences.html

[18] Juniper Networks, ‘Junos OS Architecture Overview | Junos OS | Juniper Networks’. Accessed: Mar. 25, 2025. [Online]. Available: https://www.juniper.net/documentation/us/en/software/junos/junos-overview/topics/concept/junos-software-architecture.html

[19] Juniper Networks, ‘Control Plane Distributed Denial-of-Service (DDoS) Protection Overview | Junos OS | Juniper Networks’. Accessed: Apr. 15, 2025. [Online]. Available: https://www.juniper.net/documentation/us/en/software/junos/security-services/topics/concept/subscriber-management-ddos-protection.html

[20] J. Postel, ‘RFC 768 - User Datagram Protocol’, 1980, *RFC Editor*. Accessed: Mar. 29, 2025. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc768

[21] Juniper Networks, ‘Firewall Filters Overview | Junos OS | Juniper Networks’. Accessed: Feb. 28, 2025. [Online]. Available: https://www.juniper.net/documentation/us/en/software/junos/routing-policy/topics/concept/firewall-filter-stateless-overview.html

[22] Python Software Foundation, ‘The Python Tutorial — Python 3.13.2 documentation’. Accessed: Feb. 14, 2025. [Online]. Available: https://docs.python.org/3/tutorial/index.html

[23] A. Wiggins, ‘The Twelve-Factor App’. Accessed: Mar. 30, 2025. [Online]. Available: https://12factor.net/config

[24] J. Bardin and R. Rampin, ‘scp · PyPI’. Accessed: Mar. 30, 2025. [Online]. Available: https://pypi.org/project/scp/

[25] Python Software Foundation, ‘8. Errors and Exceptions — Python 3.13.2 documentation’. Accessed: Mar. 30, 2025. [Online]. Available: https://docs.python.org/3/tutorial/errors.html

[26] Canonical, ‘Case Study: from hypervisors to LXD | Ubuntu’. Accessed: Feb. 27, 2025. [Online]. Available: https://ubuntu.com/engage/hypervisor-to-lxd-case-study

[27] Canonical, ‘How to create and configure a network - LXD documentation’. Accessed: Mar. 05, 2025. [Online]. Available: https://documentation.ubuntu.com/lxd/en/stable-5.0/howto/network_create/

[28] Canonical, ‘Bridge network - LXD documentation’. Accessed: Mar. 05, 2025. [Online]. Available: https://documentation.ubuntu.com/lxd/en/stable-5.0/reference/network_bridge/#network-bridge

[29] Postgresql, ‘PostgreSQL: Documentation: 17: 1. What Is PostgreSQL?’ Accessed: Mar. 30, 2025. [Online]. Available: https://www.postgresql.org/docs/current/intro-whatis.html

[30] U. Krishnamurthy *et al.*, ‘SQL Standards’, 2025, Accessed: Mar. 30, 2025. [Online]. Available: https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/SQL-Standards.html#GUID-BCCCFF75-D2A4-43AD-8CAF-C3C97D92AC63

[31] Postgresql, ‘PostgreSQL: About’. Accessed: Mar. 30, 2025. [Online]. Available: https://www.postgresql.org/about/

[32] Timescale, ‘Timescale Documentation | Timescale architecture for real-time analytics’. Accessed: Apr. 03, 2025. [Online]. Available: https://docs.timescale.com/about/latest/whitepaper/

[33] Grafana, ‘GitHub - grafana/grafana: The open and composable observability and data visualization platform. Visualize metrics, logs, and traces from multiple sources like Prometheus, Loki, Elasticsearch, InfluxDB, Postgres and many more.’ Accessed: Apr. 07, 2025. [Online]. Available: https://github.com/grafana/grafana

[34] Grafana, ‘Grafana OSS and Enterprise | Grafana documentation’. Accessed: Apr. 07, 2025. [Online]. Available: https://grafana.com/docs/grafana/latest/

[35] Grafana, ‘Time series | Grafana documentation’. Accessed: Apr. 08, 2025. [Online]. Available: https://grafana.com/docs/grafana/latest/panels-visualizations/visualizations/time-series/