Free Range Routing Monitoring and Anomaly Detection Tool

Bachelor's Thesis
Documentation

Semester: Spring 2025

Version: 01.00
Date: 2025-06-12

Project Team: Yannick Staedeli
Mino Petrizzo
Roman Cvijanovic

Advisor: Severin Dellsperger
Internal Co-Examiner: Olaf Zimmermann
External Co-Examiner: Thomas Graf
Project Partner: Open-Systems - Julian Klaiber

OST
Eastern Switzerland
University of Applied Sciences

School of Computer Science
OST Eastern Switzerland University of Applied Sciences

Abstract

Modern dynamic routing environments often suffer from limited visibility and diagnostic capabilities at the individual router level. This is especially true for setups based on Free Range Routing (FRR), an open-source routing suite that implements routing protocols like Open Shortest Path First (OSPF) and Border Gateway Protocol (BGP). Operators are typically forced to inspect each router's state manually via Command Line Interface (CLI), making it difficult to detect and understand anomalies such as missing or unexpected route advertisements. These issues can compromise network stability and make troubleshooting time-consuming and error-prone.

We developed FRR-MAD, a modular, open-source monitoring and anomaly detection tool tailored to OSPF in FRRouting, to address these issues. It parses and analyzes routing data, detects inconsistencies, and visualizes the router's state through an interactive text-based user interface (TUI). FRR-MAD consists of two components: the *frr-mad-analyzer*, which acts as the backend, collecting and evaluating OSPF data from FRR and the *frr-mad-tui*, a terminal-based frontend that presents the data for monitoring and troubleshooting. Features like structured route overviews, live anomaly highlighting, filtering, and Prometheus-compatible export simplify OSPF diagnostics and make routing behavior easier to understand, even in complex environments. An additional artifact that arose from this thesis is a containerlab-based development and test environment, which allows users to quickly spin up a realistic multi-router topology and test FRR-MAD without impacting production systems.

FRR-MAD enhances the monitoring and troubleshooting of OSPF routing in FRR environments by providing live anomaly detection and intuitive visualization. Its open-source nature encourages community collaboration. We invite the FRR community to adopt, test, and contribute to the project[1]. Developed in close collaboration with our project partner Open Systems, this tool is now being integrated into their production infrastructure.

[1] https://github.com/FRR-MAD/frr-mad

Management Summary

Management summary outlines the project's objective, methodology, result and achievements. Compared to the abstract it provides more clarity and detail what was explicitly achieved.

Current State

Free Range Routing (FRR) is a widely used network routing software suite. It implements a variety of different routing protocols like Open Shortest Path First (OSPF). However, the implementations are occasionally suboptimal due to software faults, which can lead to unexpected behavior in practice. As a result, the implementation of OSPF in FRR behaves sometimes inconsistently. This behavior can be seen by checking the Link State Database (LSDB) against FRR's static file configuration. The inconsistency appears as wrongly advertised Link-State Database (LSDB) entry. This issue is made worse by the fact, that there are no available open source solution to report such inconsistent behavior. To battle this behavior users of FRR are forced to create custom monitoring and observability processes to provide high availability of their services. And even if this inconsistency is spotted, it still needs to be manually verified. This can only be done on the device presenting the issue and only by using the interactive command terminal of FRR. This also highlights another issue. This issue requires an operator of such a service to directly interactive with a tool that can apply runtime changes. Advanced network engineers will have no issues correctly handling vtysh, but the same cannot be claimed for entry or medium level engineers. Therefore a monitoring solution to observe the current OSPF state is required.

Methodology

The project followed an agile development methodology based on the SCRUM framework. In close collaboration with the industry partner, multiple meetings were held to define and iteratively refine the Functional Requirements. This helped prioritize key features for development. In addition, relevant

Non-Functional Requirements were established to ensure high product quality.
To establish a solid foundation, a two-sprint elaboration phase was conducted. The goal of this phase was to evaluate existing tools and technologies that could be integrated into the project, thereby reducing development effort and focusing on value-adding components.
The subsequent development phase was the most substantial part of the project. All features were implemented and tested during this time. As the core components took shape, some initial requirements had to be revised, and previously considered integration solutions were discarded in favor of custom implementations.
The final solution comprises two distinct components: the frr-mad-analyzer, a background daemon responsible for data collection and analysis, and the frr-mad-tui, a terminal-based user interface that enables real-time interaction with the analyzer.

Achievements

The results of this bachelor's thesis are the previously mentioned frr-mad-analyzer and frr-mad-tui.

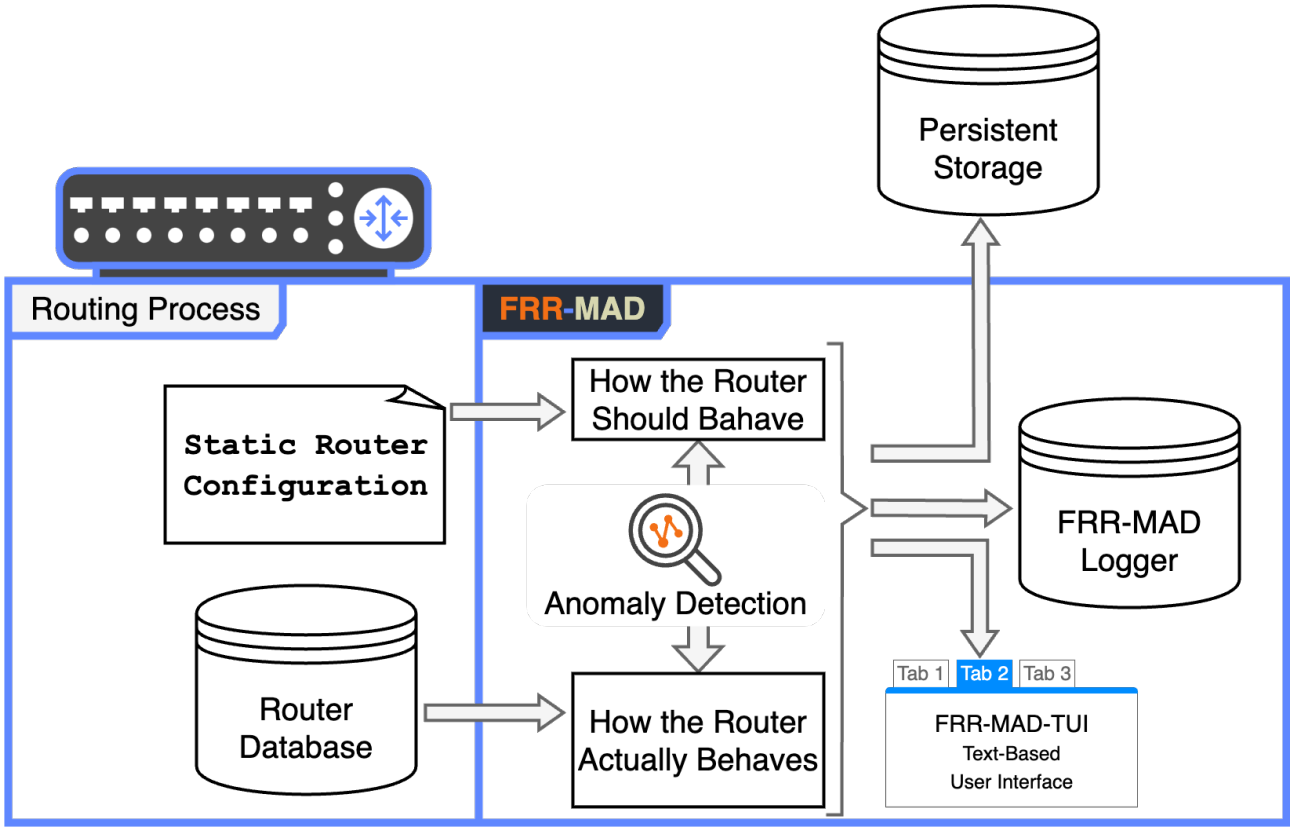


Figure 1: Simplified Architecture

The frr-mad-analyzer runs quietly in the background on a server. As long as the server is using Free Range Routing (FRR) and has routing enabled, it regularly checks two types of routing information: what the routing should look like, and what it actually looks like. It uses this comparison to find and report any problems or unexpected behavior in the network.

The frr-mad-analyzer works by creating two separate views of the network. The first view is based on the configuration and shows how the routing is expected to behave. The second view is based on the current state of the router and shows how the routing is actually working. By comparing these two views, it can quickly detect anything unusual or incorrect. If a route is found in one view but not the other, it is marked as a problem. Each issue is clearly described to help understand what is wrong and why.

In addition to finding problems, the frr-mad-analyzer also replaces an older monitoring tool used by the project partner. For this reason, it also includes built-in support to send status and performance data to external monitoring systems.

The second main part of the system is the frr-mad-tui. This is a simple, text-based interface that allows users to interact with the analyzer. It shows a clear summary of the router's current status and any problems that have been found. It also lets users explore detailed routing information in an easy-to-read format. This helps people who are not entry-level network engineers quickly understand how the router is performing.

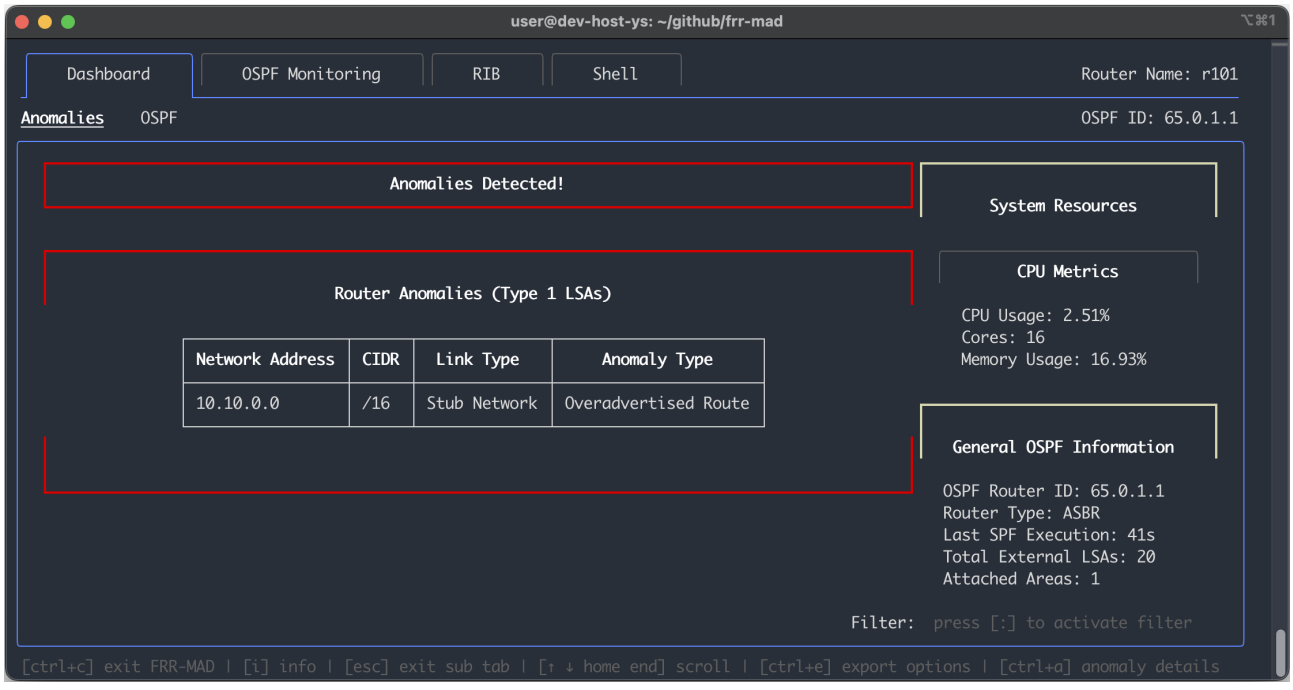


Figure 2: frr-mad-tui Anomaly Dashboard

Future Work

Both project components were released with version 1.0.0 on GitHub. But this only contains the features outlined in the requirements. During the development many experiences were gathered and some compromises were made to meet industry partner's expectations. Following that, the project will be open sourced with the organization name Free Range Routing - Monitoring and Anomaly Detection. The short term plans are:

- Adjust the anomaly detection to be compatible with the newest version of Free Range Routing.
- Enable increased modularity for the daemon component to support more standalone features. For example, a software running in the background that only provides basic monitoring data for external tools.
- Adjustment of the parsing system to support different configuration styles.

After implementing all pending shortcomings, the project will effectively enter maintenance mode. It will accept feature requests and bug reports as per the norm.

Acknowledgments

Lecture Sources

The following lecturers contributed significantly to the foundational knowledge of this thesis. Their lecture slides were used as key references throughout the project work.

- Prof. Laurent Metzger and Urs Baumann — Computer Networks 1 and 2
- Thomas Kälin, Prof. Dr. Farhad Mehta, and Stefan Kapferer - Software Engineering Practices

Special Thanks

Severin Dellsperger

As our project advisor, he played a crucial role in guiding us through the project. He provided valuable advice on both technical implementation and the academic writing of this bachelor's thesis. He played an essential role in making this project a success.

Julian Klaiber

As the representative of our industry partner, he did an outstanding job. He was always available for calls to discuss progress or assist with technical questions regarding FRR setups. He single-handedly tested our application in a real-world environment and provided valuable feedback throughout the project.

Prof. Dr. Olaf Zimmermann

As a lecturer in application architecture, he was the ideal person to consult during the elaboration phase. We are very grateful for the time he took to review and discuss our architectural designs, which greatly improved the quality of this aspect of our work.

Sandro Bolliger

Sandro Bolliger provided valuable technical insight regarding specialized OSPF use-cases. His expertise helped clarify specific aspects of OSPF protocol operations and implementation details.

Contents

Part I

Glossary and Acronyms

General Terms

aggregator The **aggregator** is a component of the frr-mad-analyzer service. It is responsible for querying vtysh correctly and parsing the received information into structured data objects. 3, 7, 29

analyzer The **analyzer** is a component of the frr-mad-analyzer service. The analyzer receives data objects from the aggregator and analyzes it for anomalous behavior. The results are stored in structured data objects and provided to the exporter and socket components. 3, 7

Batfish Batfish is an open-source tool that analyzes network configuration files to detect potential issues like routing loops, unreachable networks, and policy violations before deployment. It builds a model of your network from vendor configs (Cisco, Juniper, etc.) and runs verification queries to catch configuration errors in a safe, offline environment. 38

Border Gateway Protocol (BGP) A standardized exterior gateway protocol designed to exchange routing and reachability information among autonomous systems on the Internet. For an in-depth exploration, see BGP on Wikipedia. xxi, 59, 75

Cobra Configuration File The standardized YAML configuration file format used by the Cobra CLI framework, which serves as the central configuration mechanism for the entire FRR-MAD system. 13

Codecov Codecov is a code coverage analysis platform that integrates with your development workflow to track how much of your codebase is covered by tests. It provides detailed reports, visualizations, and pull request comments to help development teams identify untested code and maintain high code quality standards. https://about.codecov.io/. 95

containerlab A tool for orchestrating container-based networking labs, enabling easy deployment of virtual network topologies for testing and development purposes. 9

Context Mapping Context Maps describe the contact between bounded contexts and teams with a collection of patterns. The context map patterns describe a variety of perspectives like service

provisioning, model propagation or governance aspects The diversity of perspectives enables you to get a holistic overview of team and bounded context relationships (see here for more information) . 83

Docker A platform for developing, shipping, and running applications in lightweight containers. It allows consistent environments across development, testing, and production systems. 49

exporter The **exporter** is a component of the frr-mad-analyzer service. Its sole responsibility lies in exporting received information from the aggregator and analyzer components. 3, 7, 26, 29

Feature Such a Bounded Context represents a boundary around a set of functional features (user stories / use cases). For example, everything that is related to customer management in an insurance scenario: create customer, update customer, update customer address, etc.. 83

Free Range Routing (FRR) Free Range Routing is an open source routing stack that provides a suite of tools for managing IP routing. It is designed to be highly customizable and can be used to build a wide range of IP routing applications. 4

Free Range Routing - Monitoring and Anomaly Detection (FRR-MAD) The Project name of this bachelor's thesis. 3, 7, 24

FRR-MAD TUI Write Mode A privileged mode within the frr-mad-tui, activated using `Ctrl+W`, that grants the user access to execute shell commands and any vtysh command. Since this mode permits configuration changes, a safety mechanism ensures that the application can only be exited if the `running-config` matches the `startup-config`. 8, 14

frr-mad-analyzer The backend component of FRR-MAD that collects, analyzes, and exposes OSPF routing data and anomalies. iii, iv, 3, 7, 80

frr-mad-tui A Text-Based User Interface (TUI) for visualizing OSPF routing data and anomalies detected by the frr-mad-analyzer. iii, iv, 3, 8, 13, 21, 22, 35, 48, 50, 57, 80

Functional Requirements Specific features or functions the system must perform, such as processing data, providing output, or enabling user interactions. 55

iTerm2 A terminal emulator for macOS that offers technical features such as split panes, inline images, and support for OSC52 clipboard integration. 49

Label Cardinality The number of unique label combinations in a Prometheus time series. High cardinality increases memory usage and can degrade query performance. 12

Non-Functional Requirements Qualitative attributes the system should have, such as performance, reliability, usability, and maintainability. 55

OSC52 Protocol A terminal escape sequence that allows copying text to the system clipboard over SSH by encoding the data in the terminal output stream. 17, 49

socket The **socket** is a component of the frr-mad-analyzer service. It ties together the frontend and the backend. Spawning a Unix socket, it enables the frontend direct access to the various gathered and analyzed metrics. 3, 8

System The system Bounded Context allow to model a software from a more physical perspective (deployment). Examples for systems: a single page application for the frontend, a Spring Boot application that realizes the domain logic, an Oracle database that holds the data, etc. Thus, an application typically consists of multiple systems. 83

Technical Concepts

AIR - Hot Module Reloading (HMR) Air is yet another live-reloading command line utility for developing Go applications. Run air in your project root directory, leave it alone, and focus on your code. https://github.com/air-verse/air. 127

AS External LSA (Type 5 LSA) AS-external-LSAs describe routes to destinations external to the Autonomous System. AS-external-LSAs are the only type of LSAs that are flooded throughout the entire Autonomous System (except stub areas). For a comprehensive overview and deeper insights, visit RFC2328, Chapter AS-external-LSAs. 12, 13, 19, 42, 46, 132

ASBR Summary LSA (Type 4 LSA) ASBR-summary-LSAs are generated by ABRs to describe routes to Autonomous System Boundary Routers (ASBRs). These LSAs allow routers within an area to reach inter-AS destinations. See RFC2328 for more information. 13

Duplicated Route A route that is announced (advertised) multiple times with conflicting attributes, such as different costs or paths. This can cause confusion in the network, as devices may not know which route to use, potentially leading to inefficient or incorrect routing of data. 46, 62, 63

Forwarding Information Base (FIB) A lookup table used by routers to actually forward packets. It contains only the best route to each destination, selected from the Routing Information Base (RIB). 6, 14, 16

Free Range Routing - Monitoring and Anomaly Detection An open-source monitoring and issue detection toolkit for FRRouting. It consits of a daemon for static frr config and lsdb cross analysis and anomaly detection. Reporting useful information via a self-built tui or a prometheus node exporter compatible layer. iv

frr_exporter Prometheus exporter for FRR version 3.0+ that collects metrics from the FRR Unix sockets and exposes them via HTTP, ready for collecting by Prometheus. 4, 38, 39, 71

FRRMon A proprietary monitoring application developed by Open Systems for FRRouting (FRR) anomaly detection and alerting. The current system provides automated alerting capabilities but lacks a TUI/CLI interface. This project aims to replace FRRMon while maintaining its core monitoring functionality and adding interactive features. 60, 67

Interface List In OSPF context, this refers to the collection of network interfaces participating in OSPF routing, including their states, assigned areas, network types, and associated metrics. 13

Interior Gateway Protocol (IGP) A class of routing protocols used to exchange routing information within a single autonomous system. Common IGPs include OSPF, RIP, and IS-IS. 4

Link State Database (LSDB) A structured repository maintained by link-state routing protocols, such as OSPF, containing detailed topology information about all network segments and their interconnections. Routers within an OSPF area synchronize their LSDBs to ensure consistent and accurate routing decisions throughout the network. ii, xxi, 11, 13

Link State Routing Algorithm An interior routing protocol that ensures each router maintains accurate knowledge of all other routers, their links, and associated costs (metrics) within a network, stored in a shared topology database. 4

Link-State Advertisement (LSA) A fundamental data structure in OSPF used to exchange routing and topology information between routers. LSAs describe the state of router links and are distributed throughout an OSPF area to build a consistent Link State Database (LSDB). Each LSA type serves a specific purpose, representing different aspects of the network state, and is propagated to defined scopes such as individual routers, areas, or the entire autonomous system. xxi, 4, 5

mission control operator An employee of our industrial partner who works in a support role. They are typically the first to encounter a problem but may not necessarily be a specialist in the field. 37, 49, 70–73, 81

Multi-Access Network A network segment where multiple devices can communicate directly with each other over a shared medium (LAN). 4

Network LSA (Type 2 LSA) A network-LSA is generated for every transit broadcast or NBMA network by the Designated Router (DR). The network-LSA describes all the routers that are attached to the network. Visit RFC2328, Chapter Network-LSAs for details. xix, 12

NSSA External LSA (Type 7 LSA) NSSA-external-LSAs are used in Not-So-Stubby Areas (NSSAs) to describe routes to external destinations. These LSAs are similar to Type 5 LSAs but are confined to the NSSA area and are converted to Type 5 LSAs by ABRs. Reference: RFC3101. 13, 19, 46

Open Shortest Path First (OSPF) A dynamic routing protocol based on link-state technology, utilizing Dijkstra's algorithm to determine the shortest path. OSPF is widely used within autonomous systems for efficient IP routing. For a comprehensive overview and deeper insights, visit OSPF on Wikipedia. ii, xxi, 3, 59, 75

OSPF Neighbors OSPF routers form neighbor relationships with adjacent routers on the same network segment. Neighbors progress through several states (Down, Init, 2-Way, Full) before achieving full adjacency. 13

Overadvertised Route A route that is being announced (advertised) to other devices in the network but should not be. This can cause confusion or unnecessary traffic in the network, as devices may try to use a route that is not supposed to exist. 7, 19, 30, 46, 62, 63

Prometheus An open-source monitoring and alerting toolkit designed for collecting, storing, and querying time-series data. It features a powerful query language (PromQL) and integrates well with cloud-native environments. 3, 11, 12, 19, 20, 61, 67

Prometheus Node Exporter . 7, 28

Route List The complete set of routes known to an OSPF router, including intra-area, inter-area, and external routes, along with their metrics and next-hop information. 13

Router LSA (Type 1 LSA) A router originates a router-LSA for each area that it belongs to. Such an LSA describes the collected states of the router's links to the area. The LSA is flooded throughout the particular area, and no further. For a comprehensive overview and deeper insights, visit RFC2328, Chapter Router-LSAs. 5, 6, 12, 19, 40, 41, 46, 47, 123, 132

Routing Information Base (RIB) A data structure used by routers to store all received and known routing information, including multiple possible routes to the same destination. These routes may contain duplicates and are evaluated based on predefined metrics to determine the most optimal path, which is then installed to the Forwarding Information Base (FIB) for actual packet forwarding. xxii, 4, 5, 14, 16

Stub Network An OSPF network type where no router adjacencies are formed, meaning transit traffic is not allowed. It can be seen as a client network. 5

Summary LSA (Type 3 LSA) Summary-LSAs are generated by Area Border Routers (ABRs) to describe inter-area routes. These LSAs are flooded throughout the backbone area and into connected non-backbone areas. For details, see RFC2328, Chapter Summary-LSAs. 13

Transit Network An OSPF network type that connects two or more routers capable of forming adjacencies, typically represented by a Network LSA (Type 2 LSA). Visit RFC2328 for details. 47

Unadvertised Route A route that is expected to be announced (advertised) to other devices in the network but is missing. This can cause connectivity issues because other devices are unaware of the route and cannot send data to the intended destination. 7, 19, 30, 46, 62, 63

Unix Domain Socket A communication endpoint that enables efficient inter-process communication (IPC) on the same host system, often used for exchanging data between client and server applications without relying on network protocols. 8, 29, 137

unixGNU Linux foobar. 80

vtysh VTYSH is a shell for FRR daemons. It amalgamates all the CLI commands defined in each of the daemons and presents them to the user in a single shell, which saves the user from having to telnet to each of the daemons and use their individual shells. More information here. ii, 8, 14, 15, 22, 48

Wrongly Advertised Route A route that is announced (advertised) with incorrect information, such as the wrong destination or subnet mask. This can lead to data being sent to the wrong location or being unable to reach its intended destination. 46, 62, 63

Acronyms

API Application Programming Interface. 7

arch architecture. 111

BDR Backup Designated Router. 4

BGP Border Gateway Protocol (see Border Gateway Protocol (BGP)). 4, 9

CLI Command Line Interface. 80

doc documentation. 110

DR Designated Router. xviii, 4, 47

FRR Free Range Routing. ii, iii, 4, 5, 7, 10, 41, 45, 59, 98

FRR-MAD Free Range Routing - Monitoring and Anomaly Detection. 7, 9

FRs Functional Requirements. 3, 59, 93

LAN Local Area Network. xviii

LSA Link State Advertisement (see Link-State Advertisement (LSA)). 6, 61, 62

LSDB Link-State Database (see Link State Database (LSDB)). ii, 4–7, 15, 18, 38, 123

MVP Minimal Viable Product. 59, 113

NFRs Non-Functional Requirements. 69, 93

OSPF Open Shortest Path First (see Open Shortest Path First (OSPF)). ii, 4, 7–10, 40

RIB Routing Information Base (see Routing Information Base (RIB)). 5, 6, 21

SLA Servise Level Agreements. 6

TUI Text-Based User Interface. xv, 8, 13, 22, 23, 38, 80

Part II

Technical Report

Chapter 1

Introduction

This chapter introduces the context and scope of this bachelor's thesis. It begins by outlining the overall structure of the thesis, followed by a discussion of key terms and techniques used throughout the document. These terms are explained only to the extent necessary for understanding the content. Subsequently, the section on aims and objectives provides a concise overview of the core issues addressed and the goals pursued. Altogether, this chapter offers a clear starting point for what the thesis aims to achieve.

1.1 General

This thesis is structured into two main segments: a Technical Report and Project Documentation.

1.1.1 Technical Report

The Technical Report is the first part of this thesis and is divided into three chapters. The first chapter is the Introduction and familiarizes the reader with the overall context of the project. The three sections present the structure of the thesis, elaborate on key terms and techniques used, and define the aims and objectives of the work.

The second chapter of the technical report, Results, examines the results in detail. It begins by distinguishing this work from existing solutions and emphasizes the uniqueness of the presented approach. In the following section the Achievements are listed. Many objectives have been achieved and they are detailed, to demonstrate the capabilities of the solution. Finally, the chapter concludes with an in-depth analysis of the Implementation.

The final chapter of the technical report, Conclusion, evaluates the project outcomes and assesses the predefined goals. It also outlines potential directions for future improvements.

1.1.2 Project Documentation

During the four-month development period of this thesis, numerous strategic and technical decisions were made. While these decisions are grounded in technical expertise, they are not addressed in the concise technical report, as they fall outside its focused scope.

The Project Documentation begins by outlining the Requirements of the industry partner. While the Project Achievements section presents the fulfilled requirements from a user-oriented perspective, showcasing what the system delivers, the Requirements chapter documents the original Functional Requirements (FRs) as they were defined during the elaboration phase in collaboration with the industry partner. The four chapters that follow provide a structured account of the project's strategic approach. It covers tool and framework selection, architectural decisions, design rationale, quality assurance practices, and documentation standards. The Testing chapter ensures comprehensive validation of the thesis deliverables.

The concluding chapters of the project documentation address project planning, strategic development, and time management tracking.

1.2 Terms & Techniques

In this section, key concepts are presented to provide the necessary context for this thesis.

1.2.1 FRR-MAD

Free Range Routing - Monitoring and Anomaly Detection (FRR-MAD) is the name of the solution developed in this bachelor's thesis and consists of two main components, described below.

frr-mad-analyzer is an open-source Open Shortest Path First (OSPF) state analysis and information export tool. The frr-mad-analyzer is the daemon of this solution and is the first out of two tools resulting from this project. Provided Free Range Routing is running and OSPF enabled, this tool operates stand-alone. It's designed to analyze the state of an OSPF router and export the data in a format that Prometheus can understand. It consists of multiple components that act together as one service. The components are respectively: `aggregator`, `analyzer`, `exporter` and `socket`.
→ In the context of this thesis, **backend** is synonymous to **frr-mad-analyzer**.

frr-mad-tui is an open-source text-based user interface that complements the daemon frr-mad-analyzer. While the backend is a stand-alone daemon, the frr-mad-tui differs in nature. It's an extension to frr-mad-analyzer and only functions when the analyzer is running. It provides a text-based user interface to monitor any OSPF related anomalies or the OSPF and FIB state.
→ In the context of this thesis, **frontend** is synonymous to **frr-mad-tui**.

1.2.2 Free Range Routing

Free Range Routing (FRR) is an open-source IP routing protocol suite designed to provide high performance and flexible routing solutions for modern network infrastructures. It supports a wide range of routing protocols, including OSPF, BGP, RIP, IS-IS, and more, making it suitable for both enterprise and service provider environments. FRR is built on a modular architecture, allowing individual routing daemons to operate independently while sharing routing information through a central Routing Information Base (RIB).
Originally derived from the Quagga project, FRR has evolved with a strong focus on performance, scalability, and modern protocol support. It is actively maintained by a community of developers and contributors, including major networking vendors and open-source advocates.
In the context of this thesis, FRR is particularly relevant as it is the routing software suite deployed by the industry partner. Their infrastructure relies on FRR to implement OSPF, BGP, and maybe other routing functionalities, making it a critical component for understanding and evaluating the current state of the system as well as for developing improvements.

1.2.3 Open Shortest Path First

Open Shortest Path First (OSPF) is regarded as a legacy routing protocol. Its foundational specification, detailed in RFC2328, dates back to 1998. Despite its age, OSPF continues to play a significant role in contemporary networking solutions, as emphasized by the industry partner Open Systems. OSPF is a widely used Interior Gateway Protocol (IGP) based on the Link State Routing Algorithm, which dynamically distributes routing information within autonomous systems to efficiently determine optimal paths for data transmission. In such networks, OSPF routers maintain identical copies of a Link-State Database (LSDB), which is a detailed repository containing topology information about the entire network segment. To prevent excessive flooding of Link-State Advertisement (LSA) among multiple routers in Multi-Access Network environments, OSPF employs an election process to select a Designated Router (DR) and a Backup Designated Router (BDR). The DR is responsible for consolidating and distributing routing information to other routers, significantly reducing the amount of protocol-related network traffic. The BDR acts as a standby, ensuring rapid convergence and continuity of routing information exchange if the DR becomes unavailable.

1.3 Aims and Objectives

This section outlines the issues addressed in this thesis and the corresponding solutions.

1.3.1 Problem

The existing monitoring infrastructure for Free Range Routing (FRR), particularly in the context of dynamic routing protocols such as OSPF and BGP, reveals substantial limitations. Current tools, including the frr_exporter, lack the analytical depth required to ensure consistency between the static

configuration and the actual operational state of the network. Beyond insufficient analytical capabilities, there is also a notable absence of mechanisms to effectively monitor the behavior of individual routers within the network.
These shortcomings hinder the detection and diagnosis of common routing issues such as missing, unexpected, or incorrect route advertisements, as well as duplicated entries. Furthermore, existing monitoring solutions do not adequately explain discrepancies between routes advertised in Link-State Advertisement (LSA) or configured in FRR and those ultimately installed in the Routing Information Base (RIB). This diagnostic gap limits the ability to troubleshoot routing anomalies in a timely and effective manner.

Common Ruting Issues

To illustrate the routing issues consider the following router configuration:

```
!
interface eth1
    ip address 10.0.0.1/24
    ip ospf passive
    ip ospf area 0.0.0.0
exit
!
interface eth2
    ip address 10.10.10.1/24
    ip ospf passive
    ip ospf area 0.0.0.0
exit
!
router ospf
    ospf router-id 1.1.1.1
exit
!
! INFO: no other interface is configured on this router
```

Listing 1.1: Example Router Interface Config

Important to understand, that the routing issues can occur between configuration and LSDB, or between LSDB and RIB. This means inconsistencies may arise either during the translation of static configuration into Link-State Advertisement (LSA) or during the installation of routes from the LSDB into the Routing Information Base (RIB).

In case of static configuration translation, it's safe to say, that exactly these two network IDs of `eth1 (10.0.0.0/24)` and `eth2 (10.10.10.0/24)` must be advertised as Stub Network in the Router LSA (Type 1 LSA).

Issue	Example LSDB of Router LSA (Type 1 LSA)

Unadvertised
A route that is expected to be announced (advertised) to other devices in the network but is missing.
here: `10.10.10.0/24` is missing
	`Link connected to: Stub Network`
	`(Link ID) Net: 10.0.0.0`
	`(Link Data) Network Mask: 255.255.255.0`

Overadvertised
A route that is being announced (advertised) unexpectedly to other devices in the network but should not be.
here: `10.10.10.0/14` is unexpected
	`Link connected to: Stub Network`
	`(Link ID) Net: 10.0.0.0`
	`(Link Data) Network Mask: 255.255.255.0`
	`Link connected to: Stub Network`
	`(Link ID) Net: 10.10.10.0`
	`(Link Data) Network Mask: 255.255.255.0`
	`Link connected to: Stub Network`
	`(Link ID) Net: 10.10.10.0`
	`(Link Data) Network Mask: 255.252.0.0`

Duplicated Advertised
A route that is announced (advertised) multiple times with conflicting attributes, such as different costs or paths.
here: `10.10.10.0/24` is duplicated
	`Link connected to: Stub Network`
	`(Link ID) Net: 10.0.0.0`
	`(Link Data) Network Mask: 255.255.255.0`
	`Link connected to: Stub Network`
	`(Link ID) Net: 10.10.10.0`
	`(Link Data) Network Mask: 255.255.255.0`
	`Link connected to: Stub Network`
	`(Link ID) Net: 10.10.10.0`
	`(Link Data) Network Mask: 255.255.255.0`

Table 1.1: Routing Issues between Static Config and LSDB

To understand routing issues between the LSDB and the RIB, it's important to note that the RIB is allowed to contain multiple routes to the same prefix. Only the Forwarding Information Base (FIB) enforces a single best route per destination, typically the shortest path based on administrative distance and metric [6].
Therefore, verifying routing correctness requires analyzing the complete LSDB (including LSAs originated by other routers) and ensuring that every advertised prefix is present in the RIB as contributed by OSPF.

Industry Partner's Initial Situation

The industry partner operates all routers using FRR version 8.5.4. Due to the critical nature of their services and strict Servise Level Agreements (SLA), performing software upgrades (which typically require router restarts) is not feasible without risking service disruptions. As a result, they are constrained to using this specific version of FRR, which limits access to newer features, optimizations, and potential bug fixes available in later releases.

Our industry partner presented a specific problem encountered in their FRR environments, which ultimately served as the motivation for this thesis. The issue emerged during configuration changes and led to the occurrence of Unadvertised Routes and Overadvertised Routes routes. Further technical details can be found in the appendix.

Core Issues	Problem
Limited router-level OSPF monitoring: Resulting in time-consuming troubleshooting | **Missing (Unadvertised) routes** | **Unexpected (Overadvertised) routes** | **Version lock-in:** Industry partner constrained to FRR 8.5.4	

1.3.2 Solution

The challenges mentioned above highlight the need for an enhanced anomaly detection system capable of generating timely alerts, a solution to visualize the routing configuration of FRR, and a mechanism to persistently store routing metrics for further analysis and correlation. To address these needs, the novel tool Free Range Routing - Monitoring and Anomaly Detection (FRR-MAD) is introduced, providing a focused set of essential capabilities for monitoring, visualization, and anomaly detection in OSPF-based FRR environments.
Due to time constraints, it was not feasible to implement all optional features, including the BGP-related enhancements. As a result, the current version of FRR-MAD is specifically focused on OSPF anomaly detection and monitoring.
We're excited to share that our industry partner, **Open Systems**, is already using FRR-MAD in production. Our team will continue maintaining and improving the project beyond the scope of this thesis. Several ideas for future enhancements are already in the pipeline and are outlined in the Future Directions section.

Analyzer Service - frr-mad-analyzer

One of the two major achievements of this bachelor's thesis is the frr-mad-analyzer. As the name indicates, it functions as the analytical part of the solution. It comprises four distinct components, each responsible for a specific role in the data processing pipeline.

- The aggregator component retrieves FRR operational data (including running configuration, LSDB data, and routing tables) and serializes it into structured data objects for consumption by downstream processes.
- The analyzer component processes the aggregator's structured data to compute predicted LSDB states and performs comparative analysis against actual network LSDB data. Analysis outputs are encapsulated as data objects accessible to both socket and exporter component.
- The exporter component exposes metrics through a Prometheus Node Exporter-compatible endpoint, enabling standard Prometheus instances to scrape network analysis data via the Application Programming Interface (API).

- The socket component establishes a Unix Domain Socket interface, providing the frr-mad-tui with real-time access to query operational data from the frr-mad-analyzer process.

Figure 1.1 provides an overview of how the four components interact. A detailed explanation follows in the Architecture subsection of the implementation.

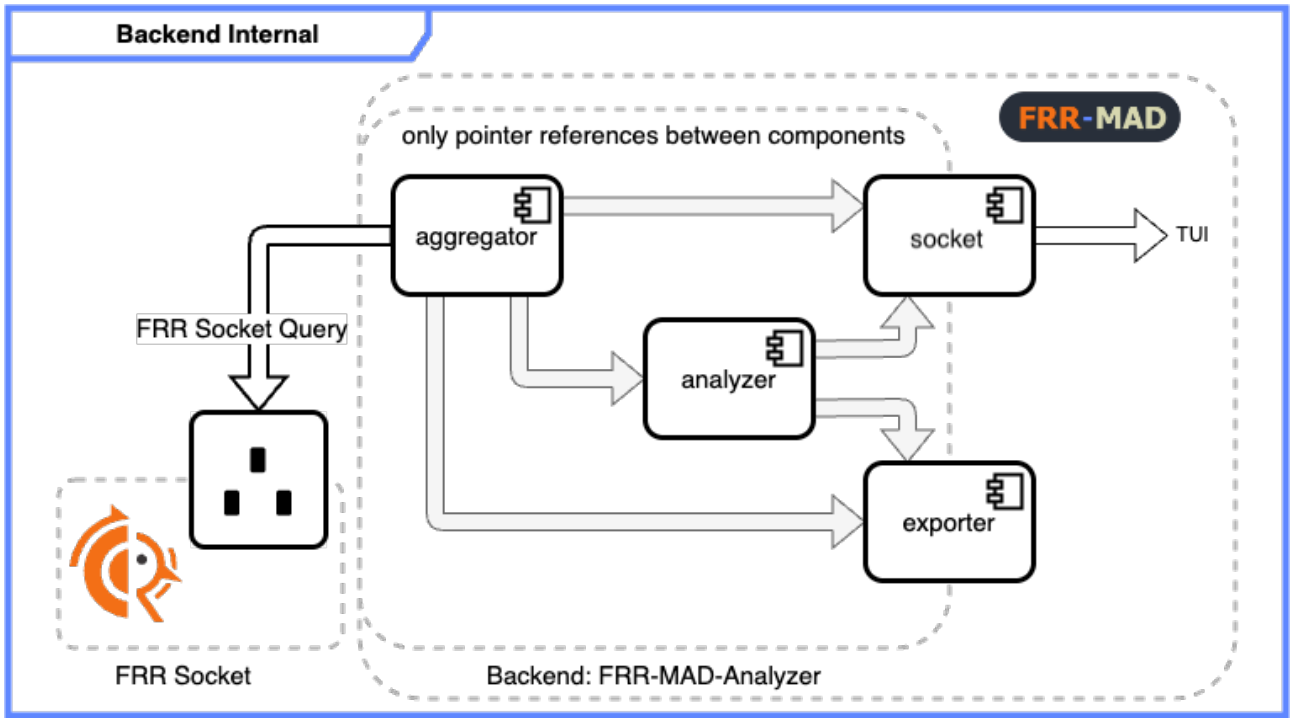


Figure 1.1: Design of frr-mad-analyzer Components

Monitoring Client - frr-mad-tui

The primary objective of this client is to provide a functional and intuitive Text-Based User Interface (TUI) that enables live monitoring of OSPF behavior and highlights any detected anomalies. It serves as a centralized interface, providing a comprehensive overview of the OSPF state on a given router. As a result, frr-mad-tui simplifies troubleshooting for network engineers and makes OSPF behavior more accessible to non-experts. Beyond its visual design (dashboard), the TUI provides practical functionality through features such as exporting parsed OSPF and FRR-related data to temporary files or directly to the system clipboard. Additionally, it supports direct execution of vtysh commands within the interface via the FRR-MAD TUI Write Mode.

The following figure demonstrate the FRR-MAD TUI, presenting the OSPF dashboard to illustrate the interface design.

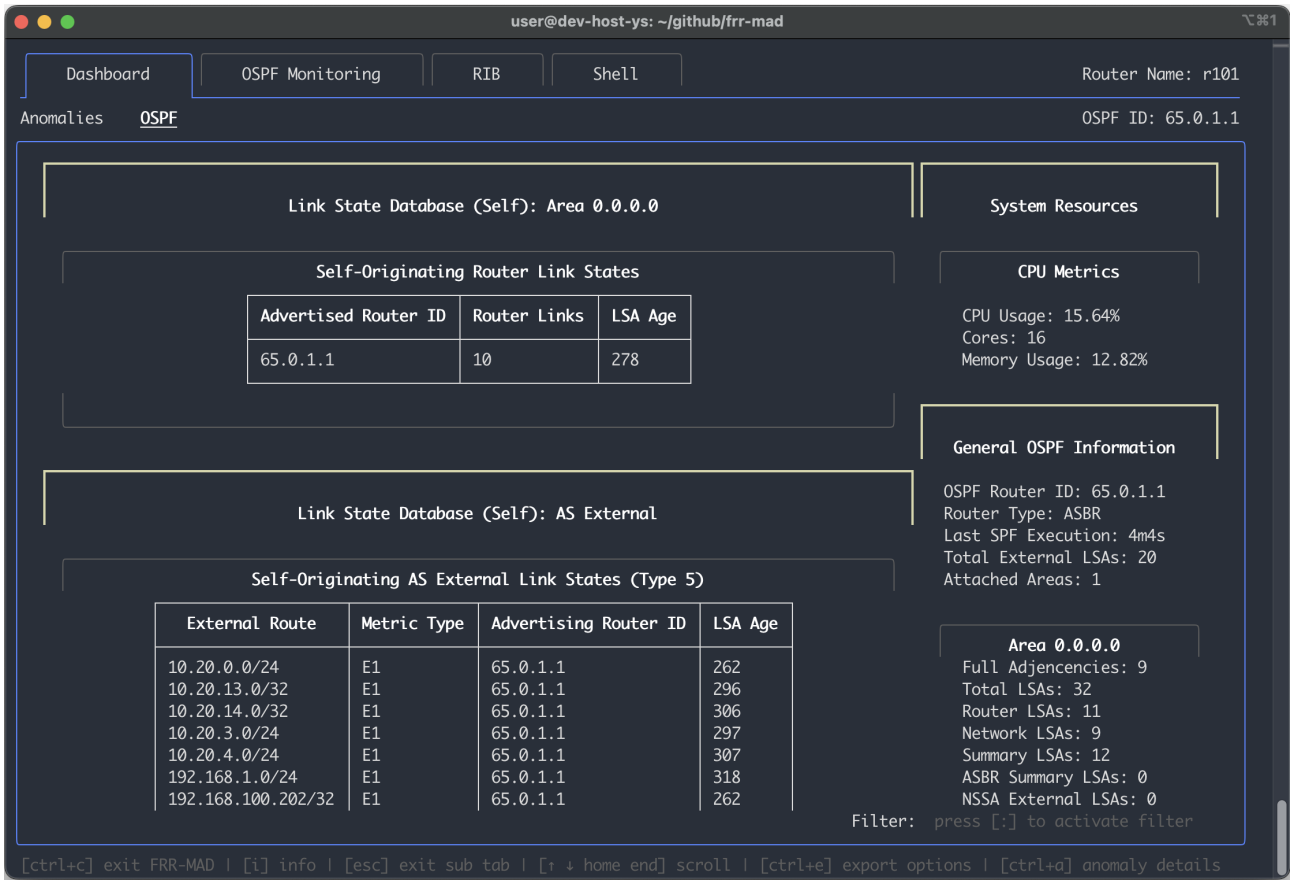


Figure 1.2: frr-mad-tui OSPF Dashboard

Development and Test Environment: Containerlab

To ensure seamless integration of all components, a significant part of this thesis focused on developing a comprehensive testing environment using containerlab.
The network topology includes multiple OSPF areas and BGP peers that interconnect distinct OSPF domains. The complexity of the topology was progressively expanded to accommodate the evolving capabilities of the thesis solution. The network concept is provided in the appendix, while the deployment files are available directly on GitHub, along with detailed documentation.
For both development and testing purposes, containerlab proved indispensable. During construction phases, the ability to rapidly test changes significantly enhanced development efficiency. Similarly, usability and acceptance testing were made feasible through this development environment.
Additionally, it serves as a ready-to-use test environment for potential users. The lab can be deployed with just a few commands, allowing anyone to quickly try out FRR-MAD in a realistic setup.

Core Artefacts	Solution
frr-mad-analyzer: Automated OSPF anomaly detection and metrics export | **frr-mad-tui:** Interactive TUI for live monitoring and troubleshooting | **Develop and Test Environment:** Containerlab-based OSPF lab for safe evaluation and demos	

Chapter 2

Results

This chapter examines the bachelor's thesis results in detail. The introduction briefly outlines the project boundaries. The two subsequent sections, Achievements and Implementation, detail the accomplished results and the final implementation respectively. The achievements section expands on the fulfilled requirements, while the implementation section describes the completed solution in comprehensive detail.

2.1 Project Boundaries

The Achievements and Implementation sections frequently reference Free Range Routing (FRR) and OSPF. It is important to clarify that no modifications to the implementation of FRR or OSPF were undertaken.

The implementations presented in this project represent original contributions developed by the three thesis authors. The FRR software suite and OSPF protocol specifications served as the foundational technologies for developing both backend and frontend system components.

This solution introduces a novel approach to monitoring and anomaly detection within OSPF network domains. At the time of the project's development, no existing tools were available that provided monitoring and analysis capabilities for OSPF behavior.

2.2 Project Achievements

This section elaborates on the solution's usage and the features achieved through its implementation. A separate FRR-MAD build and setup description is provided in the appendix.

2.2.1 Gather FRR Route Information

FRR-MAD implements a data collection process for routing information. The aggregator component receives structured JSON-formatted OSPF metrics from FRR at fixed intervals. Once parsed, this data becomes available to both the exporter and socket components. Metrics are exposed through a Prometheus-compatible endpoint for external polling via the exporter (discussed in subsection 2.2.2). Concurrently, frr-mad-tui accesses this data through the socket component. This collected routing data serves as the foundation for monitoring, anomaly detection, and subsequent analysis within the FRR-MAD system.

The following list provides a concise overview of achieved functions.

Unified OSPF Monitoring

- **Full LSDB Snapshots:** Complete Link State Database (LSDB) per area, including Router, Network, Summary, and External LSAs.
- **Neighbor States:** Real-time adjacency status (up/down), DR/BDR roles, and convergence metrics.
- **RIB/FIB Data:** Installed routes with protocol origins (OSPF/BGP/static), next hops, and metrics.

System Integration

- **FRR Configuration:** Static hostname, interface IPs, and OSPF areas from running configs.
- **Resource Metrics:** CPU/memory usage correlated with routing events.

Operational Benefits

- **No Manual Polling:** Metrics are gathered automatically at configurable intervals (default: 30s).
- **Structured Outputs:** Data is normalized for consistency across visualization (TUI) and export (Prometheus).
- **Anomaly Detection Ready:** Self-originating LSAs are pre-filtered for validation checks.

Supported Data Types

Category	Examples	Use Case
Topology	Router IDs, LSAs, neighbor states	Path validation
Routing Tables	RIB prefixes, next hops, admin distances	Route policy compliance
System Metrics	CPU load, memory usage	Capacity planning

Table 2.1: Supported FRR Data Types

Why This Matters

- **Troubleshooting:** Engineers can validate that current OSPF advertisements align with the router's configuration (e.g., passive interfaces are correctly advertised as stubs). However, the tool does not provide root cause analysis or historical correlation with specific OSPF events such as LSA floods or neighbor transitions.
- **Compliance:** Ensures that live OSPF state and advertised routes conform to intended design (e.g., redistribution rules, interface types).
- **Scalability:** Efficiently handles large-scale deployments, with normalized data structures and selective filtering (see Filter TUI content).

TL;DR **Gather FRR Route Information**

FRR-MAD gathers and structures all relevant OSPF routing data, including LSDBs, neighbor states, and the routing table, in real time. This allows for high-resolution network monitoring, streamlined troubleshooting, and accurate anomaly detection across large-scale OSPF deployments.

2.2.2 Persistently store FRR Route Information

While not directly, FRR-MAD enables persistent storage of OSPF routing metrics via Prometheus. Export behavior is configurable through runtime flags or the application's configuration file. This modularity provides flexibility for users operating large-scale networks. This level of control helps manage Label Cardinality effectively, which is especially important in Prometheus environments with many routers.

Selective Metric Export (via startup flags)

Specific OSPF metrics can be enabled, by default is disabled, via command-line flags. In this example Router LSA (Type 1 LSA) and AS External LSA (Type 5 LSA) are exported:

```
frr-mad-analyzer start --ospf-router --ospf-external
```

Provided is a list of all options available.

- `--ospf-router` (Router LSA (Type 1 LSA))
- `--ospf-network` (Network LSA (Type 2 LSA))

- `--ospf-summary` (Summary LSA (Type 3 LSA))
- `--ospf-asbr-summary` (ASBR Summary LSA (Type 4 LSA))
- `--ospf-external` (AS External LSA (Type 5 LSA))
- `--ospf-nssa-external` (NSSA External LSA (Type 7 LSA))
- `--ospf-database` (Link State Database (LSDB))
- `--ospf-neighbors` (OSPF Neighbors)
- `--interface-list` (Interface List)
- `--route-list` (Route List)

Config File Based Metric Selection

Metrics can alternatively be configured using a YAML-based configuration file (the Cobra Configuration File used throughout FRR-MAD for all configurations).

2.2.3 Visualize FRR Route Information

This subsection first explains the structure of the TUI for gaining an effective overview of the system. It then describes the FRR routing information displayed on each page.

FRR-MAD-TUI Structure

All OSPF-related routing metrics gathered and exposed by FRR-MAD can be interactively viewed in the Text-Based User Interface (TUI) (frr-mad-tui). It presents FRR behavior on a router with a clear, engineer-focused design. The TUI is fully keyboard-driven, with the footer displaying real-time hints for actions and shortcuts.
Currently, the TUI is organized into four main pages, each serving a specific function. To reduce unnecessary scrolling (NFR8), pages include sub-pages or segmented views. (see Table 2.2)

Page	Purpose
Dashboard	Offers a high-level summary of OSPF metrics, system status, and routing data. If anomalies are detected, they are prominently displayed.
OSPF Monitor	Displays detailed information about OSPF neighbors, LSAs, and interface states. Serves as the core for OSPF-specific monitoring.
RIB	Presents a structured view of the current Routing Information Base (RIB) and Forwarding Information Base (FIB), including all installed routes and their attributes.
Shell	Enables direct interaction with FRR via FRR-MAD TUI Write Mode, including execution of vtysh commands and bash commands.

Table 2.2: Purpose of each frr-mad-tui Page

The Figure 2.1 shows the info page of the TUI, which can be accessed at any time by pressing the `i` key. This provides an overview of the main menu bar with the four previously described pages as well as all available keyboard shortcuts and user interactions.
In addition to navigation options, this page highlights important interface behaviors such as TUI modes (read/write) and message types (info, warning, error). It serves as a quick reference guide to help users efficiently operate the TUI.

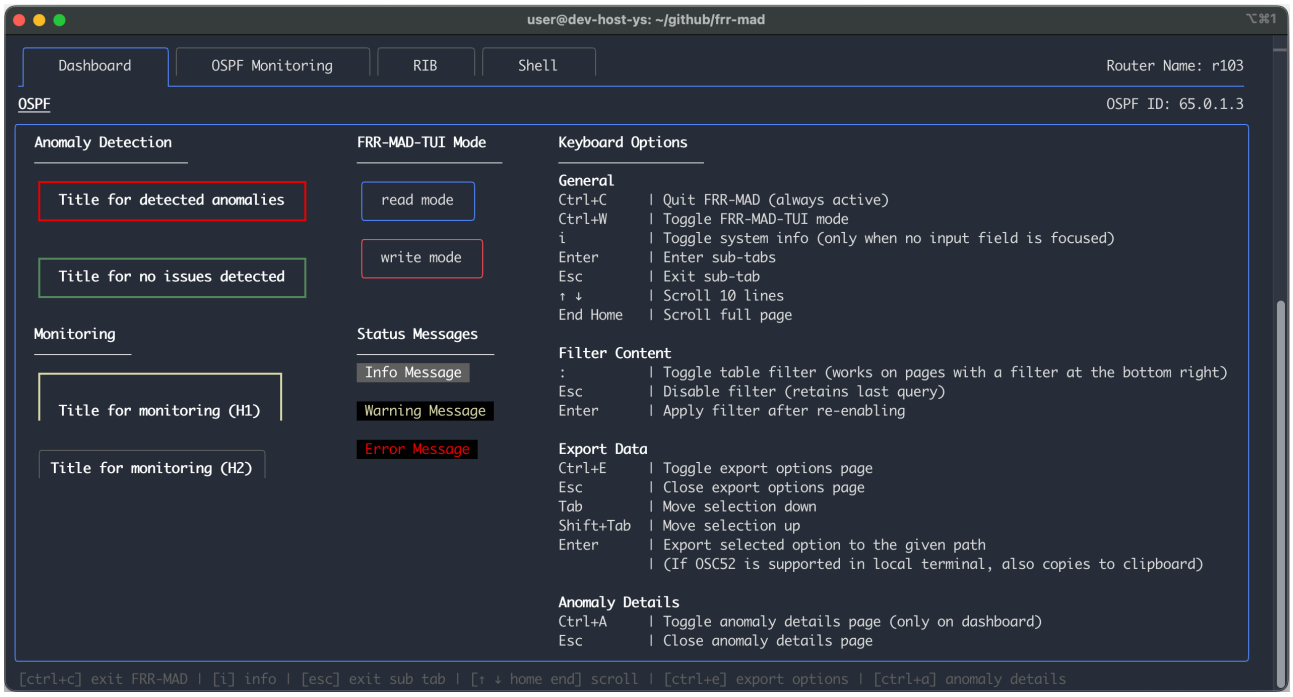


Figure 2.1: frr-mad-tui Info Page

Dashboard

The Dashboard currently contains two subpages. In the future, as more routing protocols are analyzed by FRR-MAD, each protocol will be provided with its own dedicated dashboard.

All dashboards are divided into two main panes:

- **Left Pane:** This pane is context-sensitive and updates based on the selected subpage.
  - If the anomaly dashboard is selected (default subpage of dashboards), the left pane displays anomaly-specific information.
  - The OSPF Dashboard is being used purely for monitoring, the left pane displays an overview using the vtysh command:
    `show ip ospf database self-originating`
- **Right Pane:** This provides general system information, including:
  - **Resource utilization** (e.g., CPU, memory)
  - A summary of OSPF status retrieved with the command:
    `show ip ospf`
  - Information about FRR-MAD like version and build-date

This layout ensures that both high-level system metrics and protocol-specific insights are readily available, supporting quick decision-making in both routine monitoring and anomaly scenarios.

OSPF Monitor

The OSPF Monitor page serves as the central hub for inspecting all relevant data required to monitor or troubleshoot OSPF behavior in detail. It is composed of several sub-pages, each focusing on a specific aspect of the OSPF protocol, allowing for targeted analysis and better operational insight.

- LSDB Overview: The first sub-page provides a full view of Link-State Database (LSDB), separated for each OSPF area.
  `show ip ospf database`
- Router LSA (Type 1): This sub-page presents a detailed breakdown of Router LSAs. It categorizes and displays link information in separate tables based on link type: transit networks, stub networks, and point-to-point connections.
  `show ip ospf database router self-originating`
- Network LSA (Type 2): Displays information about all Type 2 LSAs, which describe the routers connected to a shared broadcast or non-broadcast multi-access (NBMA) network.
  `show ip ospf database network self-originating`
- External LSAs (Type 5 and Type 7): This sub-page provides detailed tables for external route advertisements, which are used to represent routes redistributed into OSPF from external sources such as BGP or static routes.
  `show ip ospf database external self-originating`
  `show ip ospf database nssa-external self-originating`

- Neighbor Overview: Presents comprehensive information about all OSPF neighbors.
  `show ip ospf neighbor`
- Running Configuration: As a final sub-page, this view shows the live FRR running configuration directly within the TUI. It allows users to verify current OSPF-related settings without leaving the monitoring interface.
  `show running-config`

RIB

The RIB page provides a detailed view of the router's Routing Information Base (RIB) and Forwarding Information Base (FIB). This page is essential for understanding which routes have been learned, and which paths are ultimately used for forwarding.
It presents the following outputs:

- All known routes (prefix and next hops), their sources (e.g., OSPF, BGP, static), and associated metrics as a clearly arranged table.
  `show ip route`
- The number of routes in each routing table, which are accurately determined with the route summary command.
  `show ip route summary`

Given that routing tables can contain a large number of entries, this page integrates the TUI's filtering mechanism (see Filter Feature), which is especially useful here. Users can quickly isolate specific prefixes, next hops, or protocol entries using the interactive filter interface (activated by the `:` key).

TL;DR **Visualize Frr Route Information**

The frr-mad-tui provides a comprehensive and structured interface for inspecting OSPF routing data and system state in real time. With dedicated pages for the dashboard, OSPF monitoring, and RIB/FIB inspection, the TUI significantly enhances the visibility and manageability of routing operations. Its filtering functionality is especially valuable in large-scale environments, enabling efficient navigation of high-volume routing data. These features collectively turn the TUI into a practical and powerful tool for monitoring, troubleshooting, and understanding FRR-based OSPF networks.

2.2.4 Export Parsed FRR Route Information

The previous subsection focused on the visualization of OSPF metrics and routing data within the frr-mad-tui. This part explains how users can export the underlying parsed and aggregated data collected by the frr-mad-analyzer.

Description

The frr-mad-tui offers a convenient mechanism for exporting any currently displayed or page-specific routing data. Users can open the export menu from any page or sub-page in the TUI. The export options are context-sensitive: each page exposes exportable data relevant to its content. In most cases, this reflects the data returned by backend calls used to populate the page. In the remaining cases, direct API calls to the ospfd.vty socket provide additional information.

Two export targets are available:

- **Temporary file:** The data is written to a temporary file at a path configurable via the FRR-MAD configuration file (Listing 2.1 illustrates where to modify this path).
- **Clipboard:** When supported by the terminal, data can be copied directly to the system clipboard using the OSC52 Protocol protocol. This functionality is also available during remote SSH sessions, provided the client terminal supports OSC52.

```
default:
  tempfiles: /tmp/frr-mad
  exportpath: /tmp/frr-mad/exports # adjust this to specify tui export path
  logpath: /var/log/frr-mad
```

Listing 2.1: Adjust Export Path in Configuration File

How to

When opening the export options with `Ctrl+E` at any point in the frr-mad-tui, users can navigate through the available export targets using `Tab` and `Shift+Tab`. Pressing `Enter` triggers the export, and the user is immediately notified of the export target location. Exactly this is illustrated in Figure 2.2.

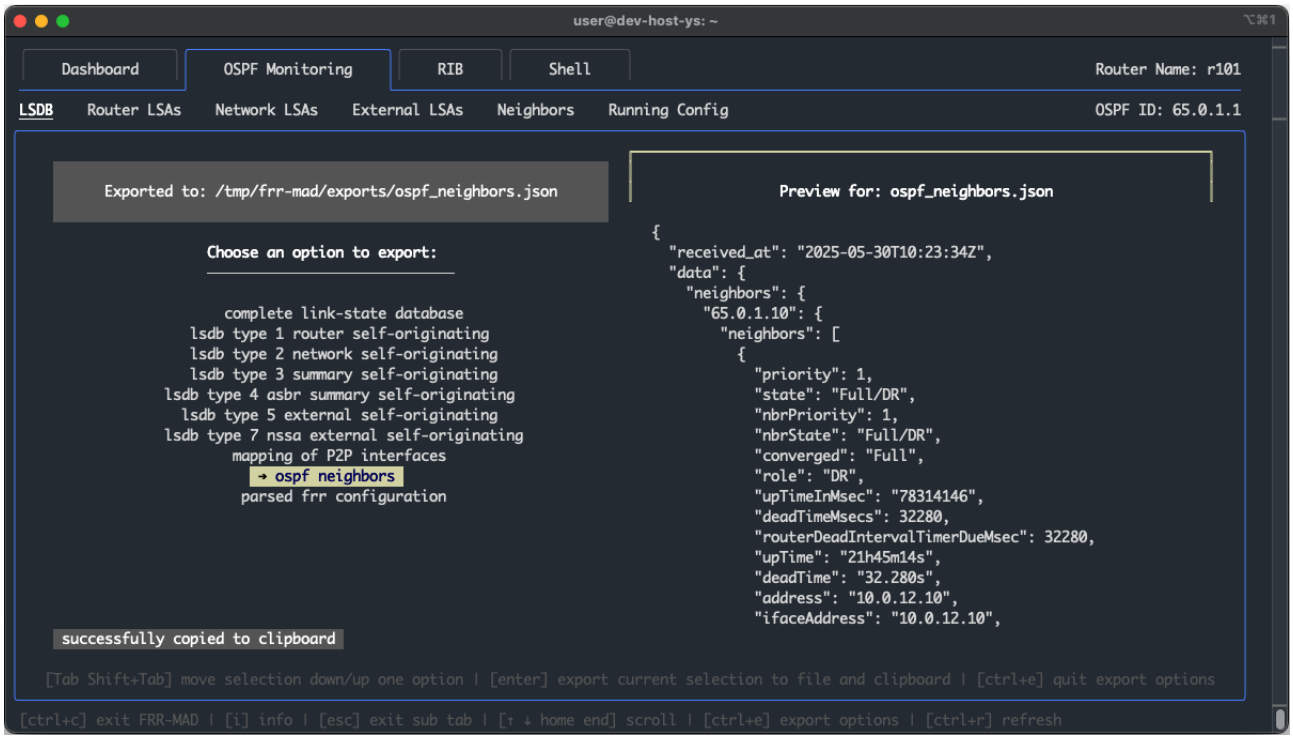


Figure 2.2: frr-mad-tui Export Page

TL;DR **Export Parsed FRR Route Information**

This export functionality enhances operational efficiency by enabling seamless transfer of routing snapshots, diagnostics, or metric outputs for further processing, sharing, or archiving. Any data parsed by the frr-mad-analyzer and used on the current page can be exported via the export menu, accessible with `Ctrl+E`. Data is exported to both the clipboard and a file located at a configurable path, as defined in the configuration file.

2.2.5 OSPF Anomaly Detection

One of the most important features is the detection of inconsistent behavior in FRR. For this purpose, frr-mad-analyzer compares two datasets to determine anomalies. The first dataset is the static FRR configuration file. Using the FRR configuration, predictions can be made for the LSDB, denoted as `shouldState`. The scope of these predictions is limited to self-originating routes only. Similarly, an `isState` is available from the router LSDB.

State Comparison

Table 2.3 presents a comparative analysis of the source metrics, designated as shouldState and isState, employed for anomalous behavior detection and analysis.

shouldState	isState	Description
FRR conf	LSDB Router	Provides the difference between the predicted Router LSA (Type 1 LSA) and the actual self-originating generated Type 1 LSDB. Contained within the same area.
FRR conf	LSDB External	Provides the difference between the predicted AS External LSA (Type 5 LSA) and the actual self-originating generated Type 5 LSDB. Exists in all areas except stub, totally stubby areas, and NSSAs, totally NSSAs.
FRR conf	LSDB NSSA-External	Provides the difference between the predicted NSSA External LSA (Type 7 LSA)and the actual self-originating generated Type 7 LSDB. Important for NSSA areas.
LSDB	FIB	Provides the difference between the populated LSDB and the actual FIB.

Table 2.3: Comparison of isState and shouldState

The concept is straightforward: compare the LSDB entries of Type 1, 5 and 7 against the configuration to identify discrepancies. The anomaly detection attempts to predict what should be added to the LSDB and then compares this prediction with the actual state.

While this approach is feasible, certain limitations must be acknowledged. The limitations will be explained in subsubsection 2.3.4

2.2.6 Export OSPF Anomalies

FRR-MAD exports all detected routing anomalies to Prometheus, providing immediate visibility into network issues without configuration overhead.

Anomaly Types Detected

As outlined in Table 2.3 there exists four different comparisons. These are further broken down into two major types of anomalies; Overadvertised Route and Unadvertised Route.

Category	Example Scenario	Severity
Over-Advertised	Static route announced in NSSA	Critical
Un-Advertised	Missing stub route	Critical

Table 2.4: Examples of Route Advertisement Issues

Operational Benefits

- **Real-Time Alerts**: Includes prebuilt Prometheus alert rules
- **Forensic Data**: Export includes:

- Source (Router/External/NSSA)
- Link-State ID
- Affected Interface
- Protocol options

Key Differences to FRR Route Information

Table 2.5 distinguishes between configurable system features and those with preset configurations. Optional data points can thus be selectively hidden, such as detailed route metrics, while anomalies are always exported.

Feature	Route Metrics (2.2.3)	Anomalies (2.2.7)
Configurable	Yes (flags/config file)	No (always on)
Storage Use	Variable (based on selected metrics)	Fixed
Alerting	Manual alert rule setup	Preconfigured alert rules

Table 2.5: Comparison of Route Metrics and Anomalies Features

Note: Both data points are available at the same Prometheus endpoint `:9091/metrics`.

2.2.7 Visualize Detected Anomalies

Figure 2.3 illustrates how an anomaly is displayed within the dashboard. All previously described anomaly types are presented in this format. For additional details on a specific anomaly, users can press `Ctrl+A` within the dashboard to open a dedicated page that provides further descriptions of each anomaly type.

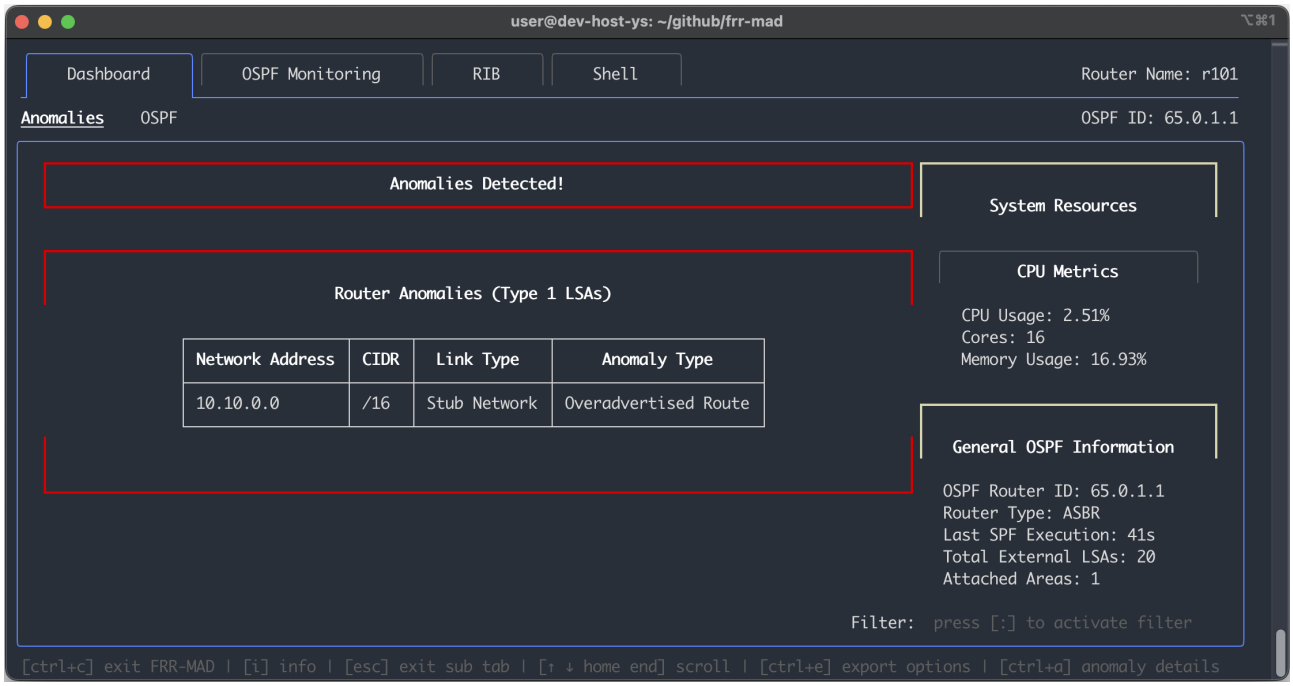


Figure 2.3: frr-mad-tui Anomaly Dashboard

2.2.8 Other TUI Features

The frr-mad-tui includes additional features that enhance the usability of both anomaly detection and OSPF monitoring.

Filter TUI Content

In large-scale network environments, such as those operated by service providers, the LSDB can grow significantly. Since OSPF does not define a strict upper limit for LSDB size, the number of entries is constrained only by the router's hardware resources. Practical deployments may contain anywhere from 1,000 to over 10,000 OSPF routes. Many vendors offer direct commands to limit the number of prefixes exported to OSPF. While FRR does not support this explicitly, similar control can be achieved using other methods[19].

To navigate and analyze such volumes efficiently, frr-mad-tui includes an interactive filtering function. This feature is essential for locating specific routes, LSAs, or configuration entries quickly. It greatly enhances usability and is particularly critical for troubleshooting and validating routing behavior in large and dynamic topologies.

As shown in Figure 2.4, the router maintains a Routing Information Base (RIB) containing 52 entries. The list is reduced to only two matching entries by applying a filter, which searches for the string `/23` marked in red.

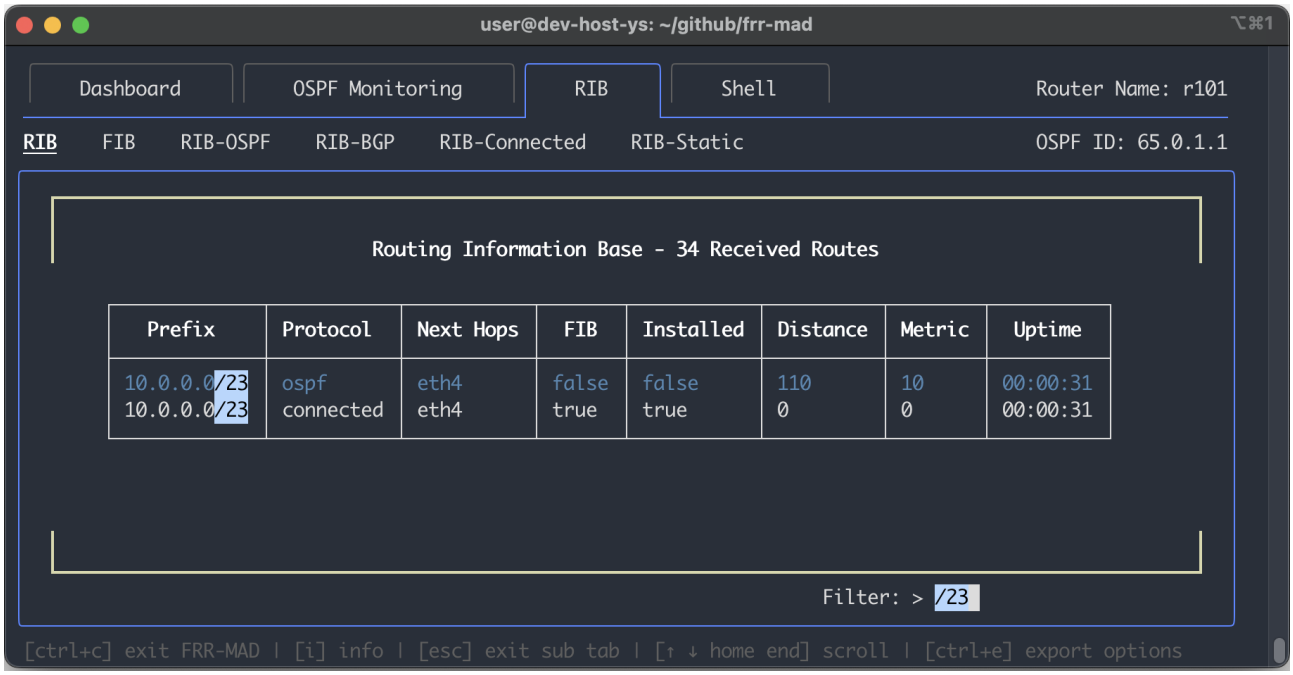


Figure 2.4: frr-mad-tui Filter Function

To activate the filter, press `:`. It can be deactivated either by pressing `:` again or `ESC`. If `ESC` is used, the current filter is retained and can be re-applied on any other page by pressing `:` again. In contrast, deactivating with `:` clears the filter immediately.

Custom Shell in TUI

The frr-mad-tui includes a custom shell interface to conduct troubleshooting workflows directly from the TUI. It consists of two subpages: a simplified shell and a vtysh interface. The simplified shell allows users to run basic bash commands such as `ip a` or `pwd`, while excluding commands that involve special characters (e.g., pipes, redirection). The second subpage provides full access to vtysh, enabling direct interaction with the FRR CLI. Figure 2.5 illustrates how the vtysh interface is used. Both shell interfaces require write mode to be enabled via `Ctrl+W`.

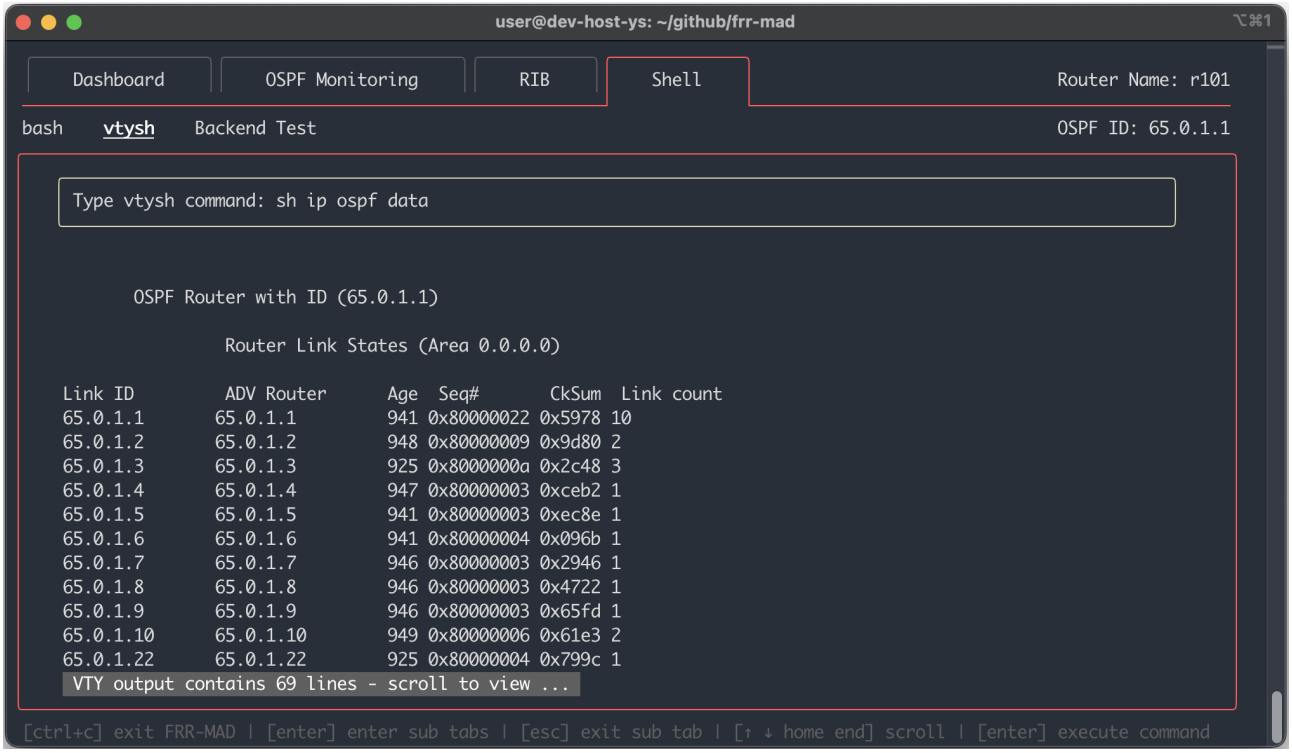


Figure 2.5: frr-mad-tui vtysh input

Status Bar TUI

The status bar is a persistent UI element located at the bottom of all TUI pages (except the running-config view). It provides real-time system feedback and status updates.
Its two primary functions are:

- **System Alerts:** Color-coded messages indicate system state, including informational updates (e.g., successful export), warnings (e.g., high resource usage), and errors (e.g., failed API calls).

  Info Message Warning Message Error Message

- **Contextual Guidance:** Dynamic hints are shown based on the current mode (e.g., read-only) or relevant actions.

The TUI automatically shortens long messages with ellipsis, based on the window size.

2.2.9 Augmented Development and Testing Environment

To conclude the achievements section, this project's development and testing environment solution is presented. To provide a system-independent test and development environment, containerlab has been utilized. Figure 2.6 illustrates the design of the development environment (see Appendix C.1 for full illustration). All routers and hosts are provided by Dockerfiles, and containerlab constructs the environment. A dedicated host was chosen to handle the build task, and through Docker mount processes, the build results are distributed across all routers.

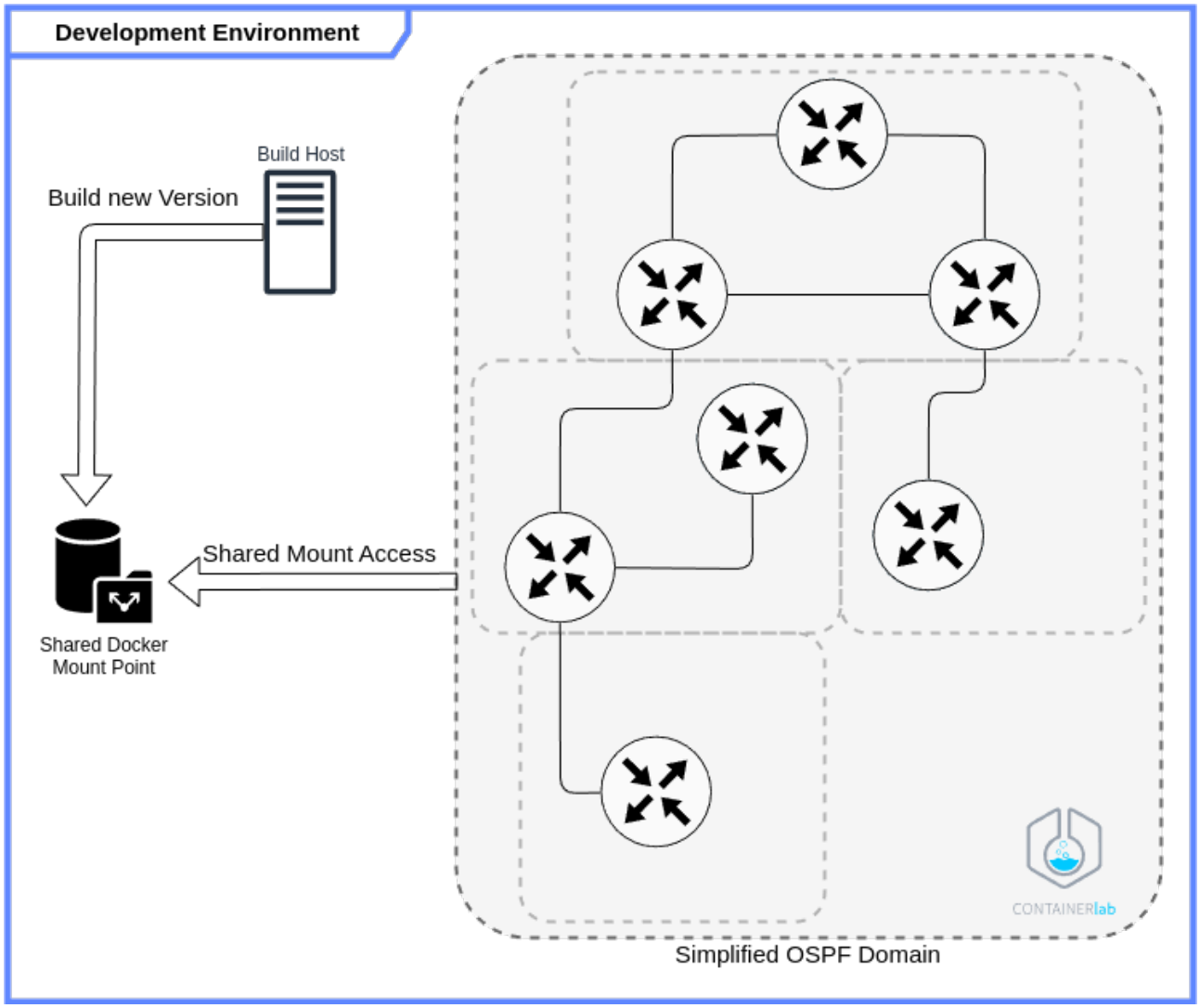


Figure 2.6: Development and Test Environment Overview

This design offers several advantages:

- System-independent development and building of files.
- Rapid replacement of build results across all containerlab devices.
- Dedicated build host reduces resource consumption compared to building on all containerlab hosts.
- Isolated build environment eliminates developer-specific build settings.
- Quick deployment and testing of the test environment.
- Enables quick, hands-on testing of FRR-MAD in a realistic setup with just a few commands.

2.3 Implementation

In the following section, the different aspects of implementing the Free Range Routing - Monitoring and Anomaly Detection (FRR-MAD) tool suit will be discussed in more detail. The overall approach

to reaching the solution will be outlined. Decisions on why certain technologies were chosen, are examined in the chapter Solution Strategy.

2.3.1 Architecture

The anomaly analysis is confined to the operation of a single device. As a result, the FRR-MAD software suite operates based on the FRR configuration and Link-State Database (LSDB) of an individual router. Figure 2.7 illustrates the interaction between the various components and services involved.

Although all illustrated entities are important, the implementation specifically focuses on the components `frr-mad-analyzer` and `frr-mad-tui`. The architectural abstraction offers a clear overview of the FRR-MAD tool suite's role within the overall system. While `frr-mad-analyzer` and `frr-mad-tui` form the core of the architecture, other referenced services and systems are included for context and to support a comprehensive understanding of the project's implementation.

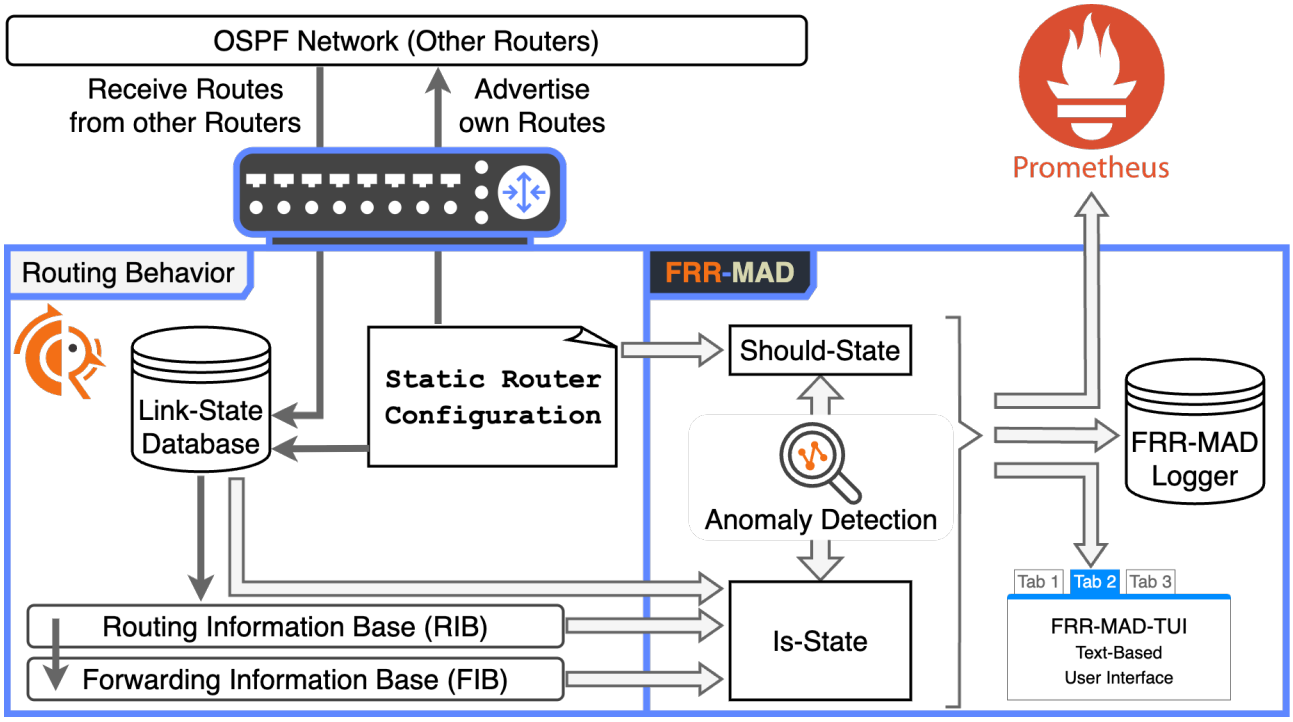


Figure 2.7: High Level Abstraction Architecture

Logger

The logging feature holds a distinct role within the FRR-MAD system. While it captures application-level behavior, it categorizes log messages into the following severity levels: `debug`, `norm`, `info`, `warning`, and `error`. In addition to general application behavior, inherent events within FRR-MAD, particularly those related to anomaly detection, are also recorded in the logs on the backend.

This logging mechanism enables more than just the use of export features from the `frr-mad-analyzer`. It allows users to directly inspect log files for any relevant state changes or anomalies. A similar ap-

proach is used for the `frr-mad-tui`, which shares the same logging foundation and stores all user interactions persistently in log files.

The log files are structured in JSON format, which facilitates efficient integration with systems that consume structured data. JSON is widely adopted due to its flexibility and readability. Each log entry includes additional metadata, most importantly, the service that generated the log and precise timestamps—ensuring traceability and consistency across the system.

Communication Channels

The communication scheme employs multiple implementation approaches, depending on the interaction layer.

FRR-MAD Suite ↔ **FRR** is the most critical communication channel. To enable effective anomaly detection, access to raw routing data is essential. FRR supports efficient querying via Unix sockets, which allows for low-latency communication using simple byte streams. This method minimizes overhead and enables rapid data retrieval.

While the use of byte-stream communication introduces certain limitations, most notably, the lack of built-in fault correction mechanisms—the performance benefits outweigh these drawbacks in the current context. The trade-off favors speed and simplicity, which are crucial for real-time analysis within the FRR-MAD system.

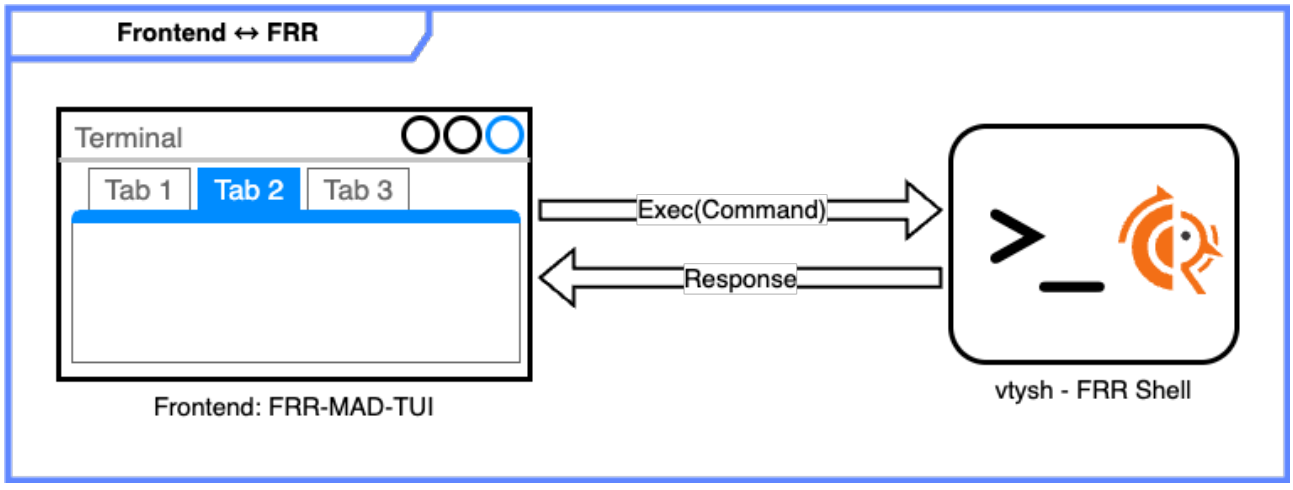


Figure 2.8: Frontend to FRR Communication

frr-mad-tui ↔ **frr-mad-analyzer** operates through decoupled communication. The frontend periodically queries the backend via Unix domain socket connections. The backend exposes a Unix socket endpoint for frontend state requests. This polling mechanism cannot reliably detect transient errors due to the absence of historical anomaly access. Two complementary systems address this limitation: the Logger and the exporter.

Both applications operate as standalone processes but require shared data object definitions. Data serialization is implemented using Protocol Buffers (protobuf) to ensure type consistency across process boundaries. Despite protobuf's native gRPC support, the implementation utilizes simple byte stream communication over the Unix socket, as shown in Figure 2.9. This design decision reflects the single-host, single-client communication pattern, which minimizes backend overhead. Given the local communication scope and fast information retrieval requirements, byte stream transmission provides sufficient performance without gRPC complexity.

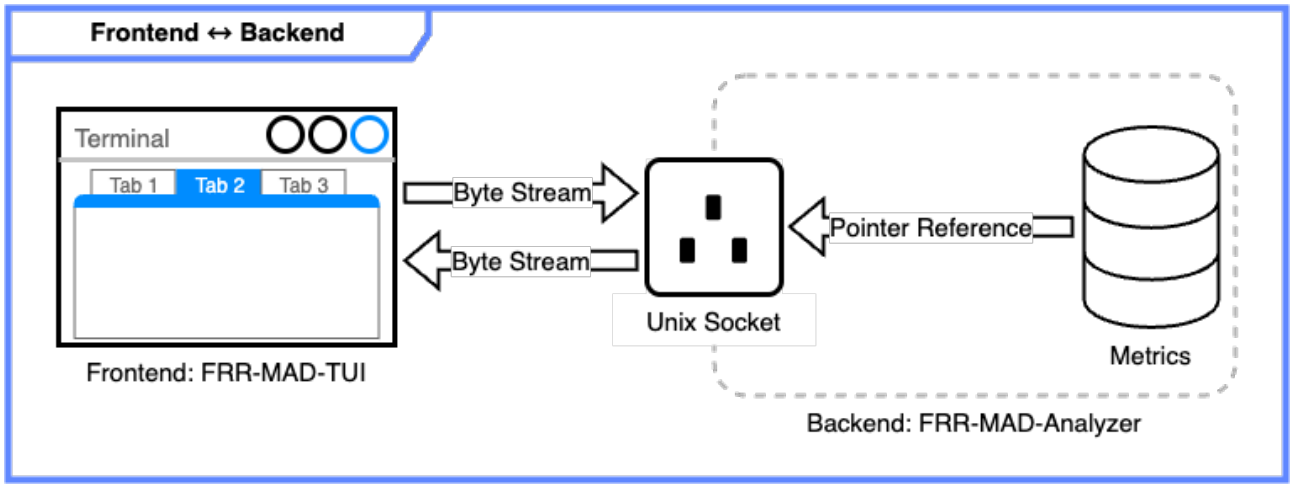


Figure 2.9: Frontend to Backend Communication

frr-mad-analyzer ↔ internal communication utilizes direct pointer references. Thanks to the use of pointer references, the application achieves a high degree of decoupling. The pointer references can be regarded as dependency injection. It provides the benefits that arbitrary data can be injected for testing purposes too. Once again protobuf is used for data serialization and handling.

The frr-mad-analyzer comprises four components: aggregator, analyzer, exporter, and communication handler. These components function as self-contained microservices with decoupled operational logic. Components expose pointers to protobuf-serialized data structures to maintain this architectural separation. An internal timer periodically refreshes data objects, ensuring all components access current metrics efficiently.

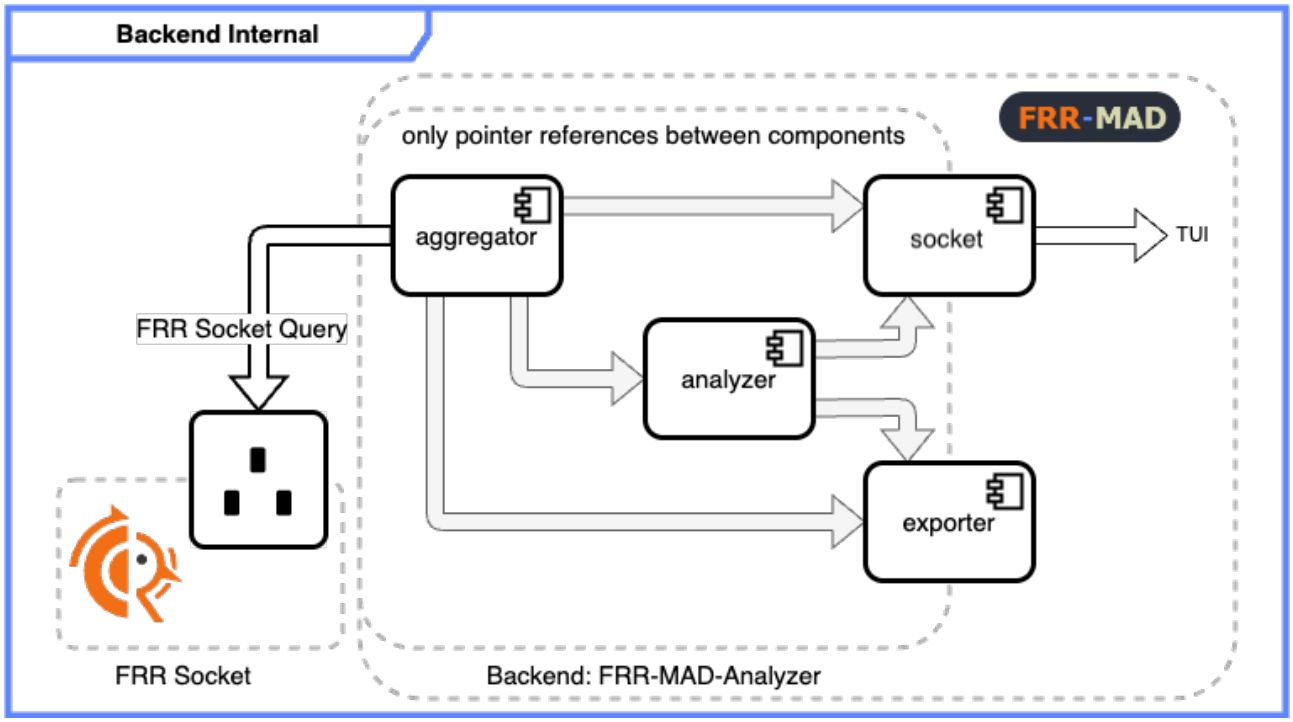


Figure 2.10: Backend internal communication

frr-mad-analyzer ↔ **Prometheus** implements the external metrics interface. This communication layer adheres to Prometheus Node Exporter endpoint specifications, as detailed in previous sections. As in internal communication explained, the metrics are made available with pointer references. The exporter exposes a copy of these metrics on a Prometheus compatible endpoint.

Figure 2.11 illustrates the communication structure. The Prometheus Node Exporter endpoint binds to all available network interfaces, enabling metric access via the configured port. The port number is configurable through the application configuration file.

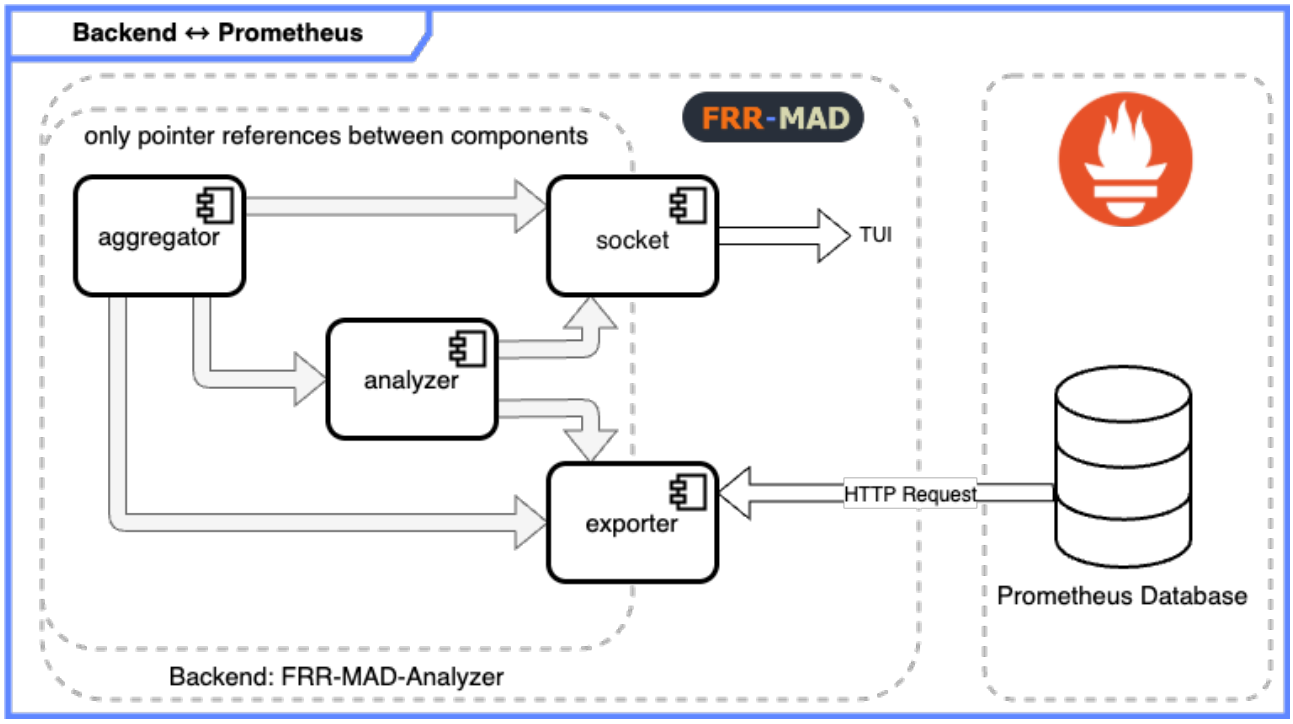


Figure 2.11: Backend - Prometheus Communication

2.3.2 Backend Service

The backend implements multiple communication paradigms. Direct synchronous communication with the frontend is established through a Unix Domain Socket interface. The exporter component facilitates the extraction and transmission of operational metrics from FRR services.
Prior to metric exportation, data collection and analysis must be performed. The aggregator component handles the retrieval and parsing of FRR operational data. The implementation follows a direct approach: information is obtained via FRR control sockets `zebra.vty` and `ospfd.vty`. Retrieved data undergoes parsing and serialization into structured data objects, which are subsequently made available to downstream components.

The following of this subsection examines the implementation of the analyzer, communications, and exporter components in detail. Refer to Section 2.2.5 to review the fundamentals of OSPF anomaly analysis.

Backend: Analyzer

The analysis process begins by generating a predicted Link State Database (LSDB) and comparing it against the actual LSDB. The prediction is derived by analyzing the FRR configuration and constructing the potential network states. This approach has an inherent limitation: only self-originating routes can be predicted using this method. Four distinct datasets are analyzed to identify potential anomalies. These datasets are calculated for LSA Types 1, 5, and 7, as well as the Forwarding Information Base (FIB).

Algorithm 1 provides an overview of the data preparation process. The analyzer receives parsed configuration data and different LSDB types as input. These two datasets have fundamentally different structures and must first be normalized to enable comparison. The most straightforward approach involves calculating the network addresses from the respective datasets and storing them in maps.

Algorithm 1: Convert OSPF Link States to Network Addresses Map

Input: $linkStateList$ (collection of OSPF link states)

Output: $networkMap$ (map of network addresses in CIDR notation with linkState information)

$networkMap \leftarrow [string]linkState$;

foreach $linkState \in linkStates$ **do**

$tmpLinkState := \{$

$ip \leftarrow$ EXTRACTIP$(linkState)$;

$mask \leftarrow$ EXTRACTMASK$(linkState)$;

$network \leftarrow$ CALCULATENETWORK$(ip, mask)$;

$cidr \leftarrow$ CONVERTTOCIDR$(network, mask)$;

$networkSet \leftarrow networkSet \cup \{cidr\}$;

$\}\ networkMap[network] = tmpLinkState$;

end

return $networkMap$

The use of maps makes checking for the existence of link states straightforward. Algorithm 2 provides an overview of how the test is performed. If an entry from the `shouldState` is not present in the `isState`, an anomaly can be inferred and is classified as Unadvertised Route.

Algorithm 2: Detect Unadvertised Networks

Input: $shouldNetworkMap$ (expected network addresses)

Input: $isNetworkMap$ (actual network addresses)

Input: $unadvertisedList$ (pointer to unadvertised networks list)

foreach $network \in shouldNetworkMap$ **do**

if $network \notin isNetworkmap$ **then**

$*unadvertisedList \leftarrow *unadvertisedList \cup \{network\}$;

end

end

Similar to the previous algorithm, Algorithm 3 abstracts the detection of Overadvertised Routes. The same logic applies, but in reverse. If a network from isNetworkMap is not present in shouldNetworkMap, it can be classified as overadvertised.

Algorithm 3: Detect Overadvertised Networks

Input: $shouldNetworkMap$ (expected network addresses)
Input: $isNetworkMap$ (actual network addresses)
Input: $overadvertisedList$ (pointer to overadvertised networks list)

foreach $network \in isNetworkMap$ **do**
  if $network \notin shouldNetworkMap$ **then**
    $*overadvertisedList \leftarrow *overadvertisedList \cup \{network\}$;
  end
end

The algorithms are executed sequentially. During the initialization of the different components, serialized data structures are created and shared between components. The information sharing process is therefore accomplished through pointer references, as previously mentioned. It is important to note that no two instances perform write operations on the same pointer reference. Only one component populates the pointer references while all other instances function as read-only consumers.

This testing procedure is generally sound, with one primary limitation: reduced accuracy. Different network types, such as stub or point-to-point networks, exhibit distinct behaviors.

While the algorithms presented above are high-level abstractions of the actual implementations, they accurately represent the general procedure. Ideally, the analyzer should infer the correct network type from the configuration, but this is not feasible with the current analyzer architecture.

Backend: Socket

As previously mentioned, the socket is implemented using a Unix socket. Byte streams transport protobuf-serialized data structures between components. Request messages are of type Message and contain the fields `Service` and `Command`. Algorithm 4 illustrates the general operation of the message handler. The socket receives a protobuf-serialized byte stream of type Message containing two data fields: `Service`, which determines the appropriate service handler for command routing, and `Command`, which specifies the requested information retrieval operation.

Algorithm 4: Unix Socket Message Handler

Input: $request \leftarrow Message\{\}$ (incoming bytestream; protobuf serialized data)
Output: $response \leftarrow Response\{\}$ (outgoing bytestream; protobuf serialized Response)

$message \leftarrow$ DESERIALIZEPROTOBUF$(request,$ Message$)$;
$service \leftarrow message$.Service;

switch $service$ **do**
  case *"ServiceA"* **do**
    $responseObj \leftarrow$ FORWARDTOSERVICEHANDLER$(\text{"}ServiceA\text{"}, message)$;
  end
  case *"ServiceB"* **do**
    $responseObj \leftarrow$ FORWARDTOSERVICEHANDLER$(\text{"}ServiceB\text{"}, message)$;
  end
  case *Default* **do**
    $responseObj \leftarrow$ CREATEERRORRESPONSE$(\text{"}InvalidService\text{"})$;
  end
end

$response \leftarrow$ SERIALIZEPROTOBUF$(responseObj,$ Response$)$;

return $response$

Upon successful service identification, the corresponding service handler processes the `Command`. Algorithm 5 presents the generic implementation of the service handler. While each service handler follows a similar structure, their specific implementations do not allow for abstraction. However, this generic algorithm suffices for explaining the common service handler pattern. Once the message is properly parsed, requests are forwarded to their respective tasks. All subtasks consistently return Response objects as a design consideration to reduce complexity at the frontend-backend socket interface. The response object either contains a dataset corresponding to the Message request or returns `nil`. In addition, it reduces the complexity of testing.

Algorithm 5: Generic Service Handler

Input: $message$ (deserialized Message object)

Output: $response$ (outgoing byte stream; protobuf serialized Response)

$command \leftarrow message$.Command;

switch $command$ **do**

  case *"CommandA"* **do**

    $result \leftarrow$ EXECUTECOMMAND$("CommandA", message)$;

    $responseObj \leftarrow$ CREATESUCCESSRESPONSE$(result)$;

  end

  case *"CommandB"* **do**

    $result \leftarrow$ EXECUTECOMMAND$("CommandB", message)$;

    $responseObj \leftarrow$ CREATESUCCESSRESPONSE$(result)$;

  end

  case *Default* **do**

    $responseObj \leftarrow$ CREATEERRORRESPONSE$("InvalidCommand")$;

  end

end

$response \leftarrow$ SERIALIZEPROTOBUF$(responseObj,$ Response$)$;

return $response$

The communication system between these two components is designed to be synchronous. In the current iteration of FRR-MAD, there is no requirement for more complex capabilities. All information accessible to the frontend is also exported through a Prometheus-compatible endpoint.

Backend: Exporter

The exporter component serves as the metrics interface between FRR-MAD and external monitoring systems. It implements a Prometheus-compatible endpoint that exposes both operational metrics and anomaly detection results. The exporter follows a modular design with two primary subcomponents: the MetricExporter for FRR operational data and the AnomalyExporter for anomaly detection results.

Metrics Export

The MetricExporter takes the data from the pointer and exposes FRR operational metrics through a configurable HTTP endpoint. Key characteristics include:

- **Thread-safe Updates**: A read-write mutex protects metric updates during concurrent access
- **Comprehensive Coverage**: Supports exporting metrics for all LSA types including OSPF neighbor, Interface status and Routing table metrics

Algorithm 6 illustrates the metric update procedure. The exporter periodically refreshes all enabled metrics from the shared data structure while maintaining atomic operations through mutex protec-

tion. This algorithm uses a generic type parameter as a placeholder for any metric type that can be exported, with specific metrics outlined in Persistently Store FRR Route Information.

Algorithm 6: Metric Export Procedure

Input: $data$ (shared FRR operational data)
Input: $config$ (exporter configuration)

```
foreach metric category do
    if category is enabled then
        switch category do
            case generic do
                Update generic metrics;
            end
        end
    end
end
```

Anomaly Export

The AnomalyExporter provides detailed visibility into detected anomalies through three metric types:

- **Binary Flags**: Indicate presence/absence of anomaly types per detection source
- **Counters**: Track total anomalies detected per category
- **Detailed Metrics**: Provide labeled instances of specific anomalies

The exporter handles four anomaly sources:

- Router LSA anomalies (Type 1)
- External LSA anomalies (Type 5)
- NSSA LSA anomalies (Type 7)
- LSDB-to-RIB inconsistencies

Algorithm 7 shows the anomaly metric update process. The exporter first resets all metrics to zero before repopulating them with current anomaly data, ensuring metric staleness is prevented.

Algorithm 7: Anomaly Export Procedure

Input: $anomalies$ (shared anomaly detection results)

Reset all anomaly metrics to zero;

foreach *anomaly source* **do**
    Update binary flags for detected anomaly types;
    Update counters for anomaly counts;
    foreach *detected anomaly* **do**
        Create detailed metric with labels;
    end
end

Implementation Details

The exporter component implements several reliability features:

- **Retry Mechanism**: Failed metric updates automatically retry once before logging errors
- **Panic Recovery**: Catches and logs panics during metric updates
- **Atomic Operations**: Uses mutexes to prevent concurrent write conflicts

The HTTP server exposes metrics on a configurable port (default: 9091) with these endpoints:

- `/metrics` - Prometheus-formatted metrics
- `/` - Simple HTML status page

2.3.3 Frontend Client

The frr-mad-tui serves as the main interaction point for users. It is built using Bubbletea, a robust and flexible Go framework for creating rich TUIs, and styled with Lipgloss, which provides composable terminal styling utilities.

Frontend Structure

The source code for the TUI is organized for scalability, following a clean separation of concerns:

```
root/
+- src/
   +- frontend/
      +- cmd/
      |  +- tui/              # Entry point for the TUI application (main.go)
      +- internal/
         +- common/           # Shared types, utilities, and helper functions
         +- pages/            # Page-based structure,
         |  +- examplePage/
```

```
10          |     +- model/       # Bubbletea model definition
11          |     +- update/      # Update logic and message handling
12          |     +- view/        # View rendering and data aggregation
13          +- services/          # Interfaces to the backend (e.g., socket
                communication)
14          +- ui/                # Shared Lipgloss-based UI components and
                layout styling
```

Listing 2.2: Frontend Structure

Design Philosophy

Each page within the TUI (e.g., Dashboard, OSPF Monitor) is implemented as an independent Bubbletea model, following a standardized pattern: model, update, and view. This structure promotes modularity and makes it easy to extend the TUI with new pages or features.
Backend interactions are handled through the *services* layer, which abstracts the protobuf communication with the frr-mad-analyzer.
Styling and layout components are centralized in the *ui* folder, which ensures consistency across all pages and enables quick design adjustments via Lipgloss.

Development Process

To become familiar with the framework and libraries, a minimal prototype was developed early in the project. While prior experience with CSS can be helpful, working with Lipgloss, and TUIs in general, presents its own unique challenges. One of the most time-consuming aspects was implementing responsive layout behavior. Unlike web development, Lipgloss does not offer built-in layout systems such as Flexbox or CSS Grid. Instead, all sizing must be handled manually in code. Functionality like `justify-content: space-between;` quickly loses its charm when you have to manually calculate the widths of container A and B, then insert a third invisible box as a spacer, just to make sure that A sticks to the left edge of the terminal and B hugs the right. Consequently, the *style.go* file contains extensive logic for calculating dynamic widths and heights, supporting various terminal segmentation (e.g., half, one-third, one-fourth).

Keywords	Frontend Client
Design Pattern: Model, View, Update | Bubbletea (Go Framework) | Lipgloss Styling | Modular Page Structure | Responsive Layout | Unix Socket Communication via Protobuf	

2.3.4 Process and Challenges

This final subsection examines the key decisions and challenges encountered during the project development phase. The first section discusses the general workflow processes and design decisions made, along with subsequent adjustments implemented during development. The second section addresses the challenges and limitations encountered in anomaly detection implementation.

Workflow

This section outlines the workflow followed by a mission control operator when facing a routing problem. The corresponding flowchart serves as a foundational reference for later project phases, particularly for the development of the TUI-MVP.

To better understand the flowchart, several key aspects should be noted:

- Data export can be made at any point of the workflow.
- The "Anomaly Detection" process contains the anomalies according to FR3.

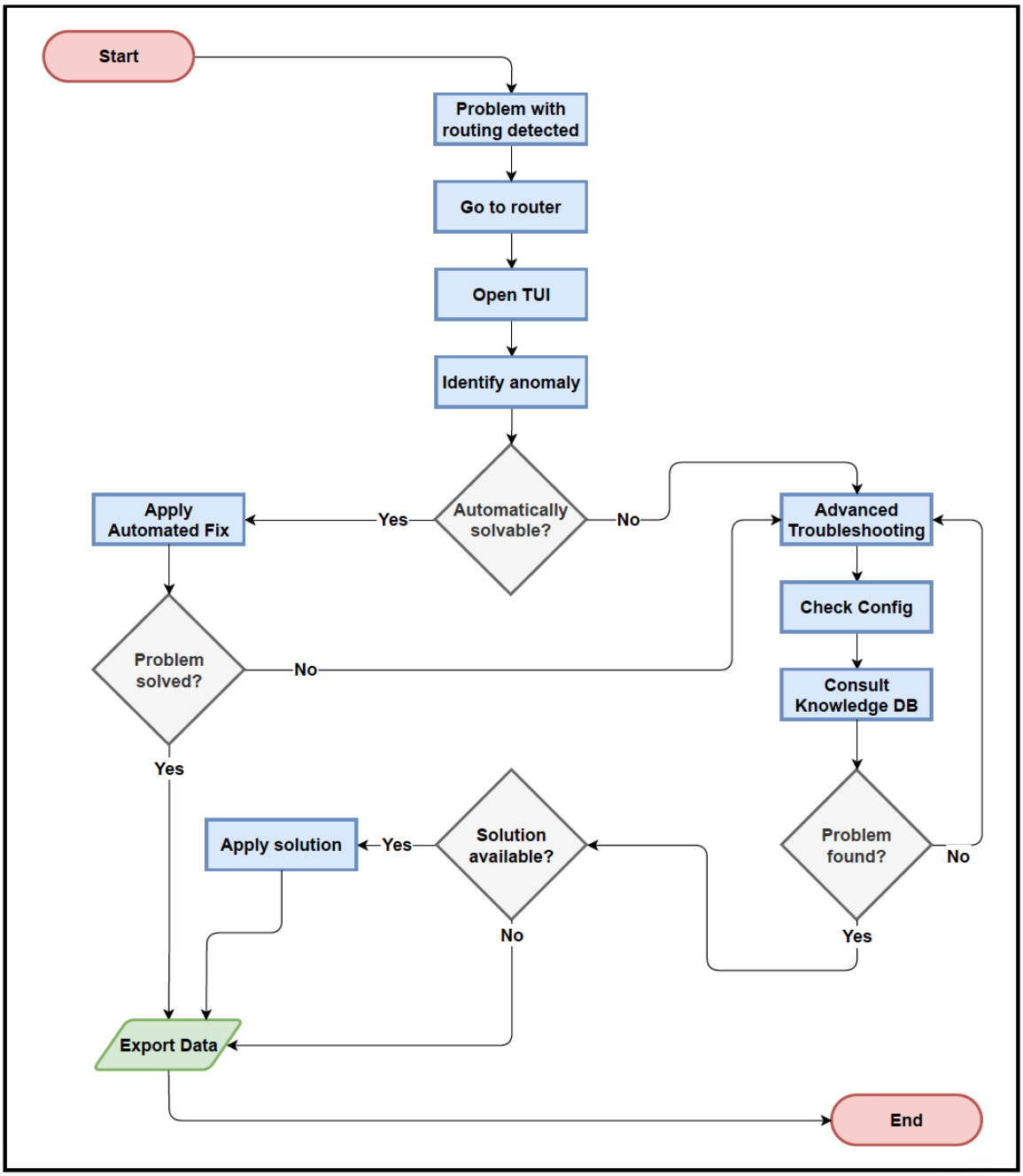


Figure 2.12: Workflow Flowchart

System Design Adjustments

The system design challenges encountered during development necessitated significant architectural modifications. Both Batfish and frr_exporter proved impractical for FRR-MAD development requirements.

Initial Design:

Figure 2.13 presents the initial system architecture following the project's elaboration phase. Initially, Batfish was evaluated as a potential engine for parsing and validating routing configurations of the complete OSPF network. Additionally, the frr_exporter was intended to handle all FRR-MAD-related analysis results export functionality.

By querying both frr_exporter and Batfish, it would obtain insights into network configuration and LSDB state. With this information the analysis process could be initiated as described in previous chapters. Finally, the frr_exporter was to be extended to export analysis results, while a Text-Based User Interface (TUI) would offer insights into the local router's state directly on the device.

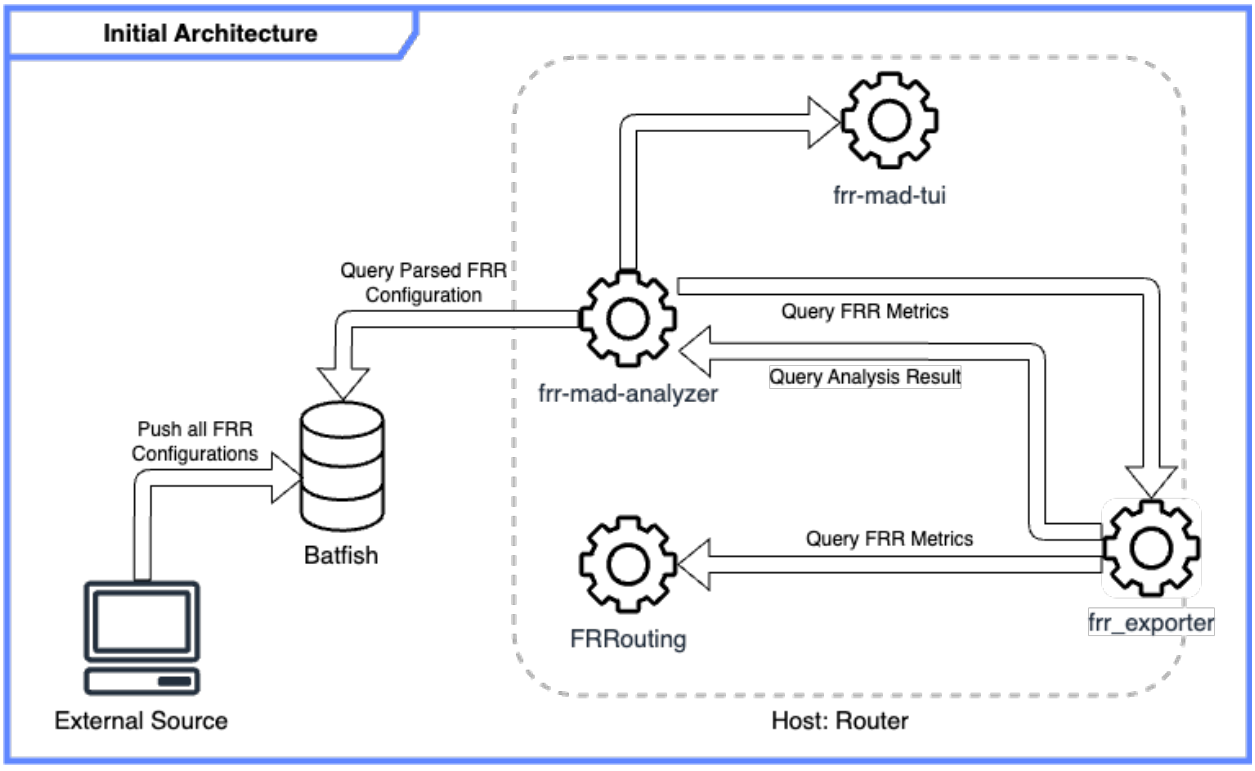


Figure 2.13: Elaboration Result: Initial Architecture

Adjusted Design:

Early in the development phase, it became evident that the above approach was insufficient. Both Batfish and frr_exporter proved to be suboptimal choices for implementation or adaptation.

Batfish required a dedicated host to perform network-wide FRR configuration validation within OSPF domains. This approach introduced practical limitations, as it would force users of the FRR-MAD toolset to deploy and maintain a centralized Batfish instance. This was neither desired by the industry partner nor aligned with our design goal to keep the solution self-contained and free of heavyweight dependencies.

The intended modifications to frr_exporter would violate its intended design principles. The frr_exporter is specifically designed to export FRR-based information without modification. The changes to frr_exporter by this project, would also see a strong coupling between the FRR-MAD toolset and frr_exporter. Based on these findings, the design shown in Figure 2.14 was chosen.

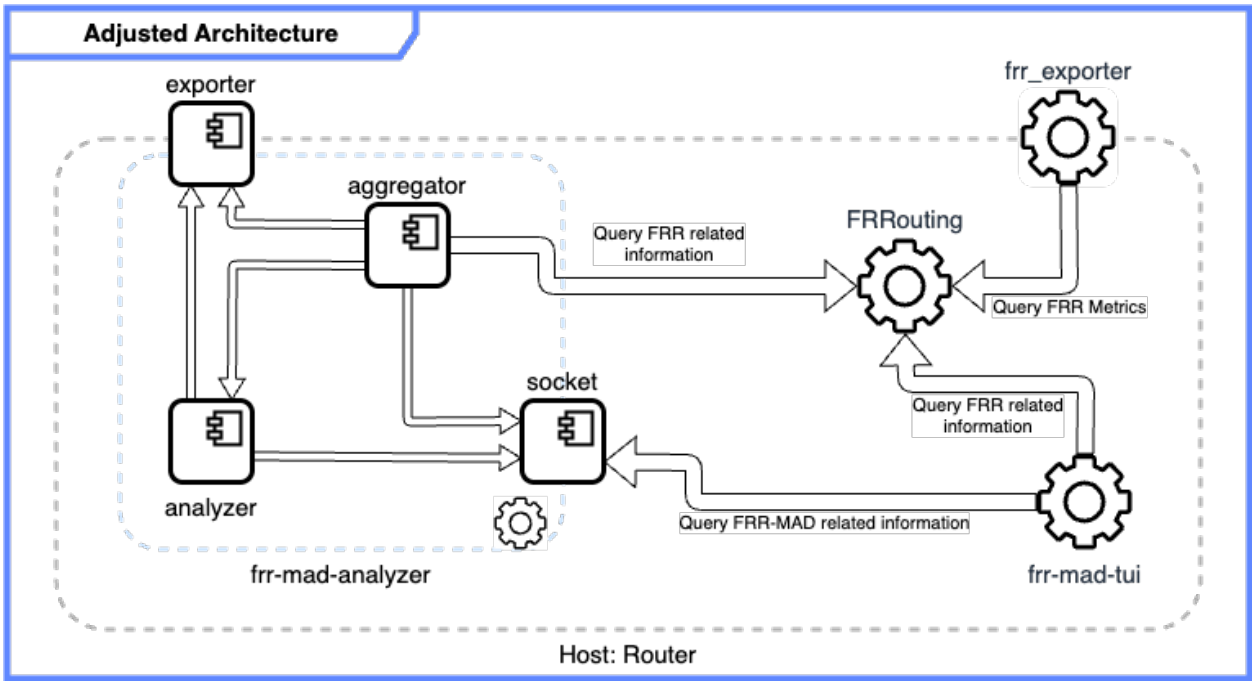


Figure 2.14: Development Phase: Adjusted Architecture

Anomaly Detection

The implementation of the analyzer logic required close collaboration with our industry partner, who provided essential domain knowledge. Since FRR-MAD is intended to run in this specific context, it was crucial to follow an agile development approach with rapid feedback loops. Despite extensive testing in our own lab environment (see Figure C.1), several issues only emerged in the industry partner's setup. These discrepancies highlighted two major challenges that arose from both a gap in our OSPF expertise and an initially limited understanding of the partner's infrastructure.
To clearly present the two problems, the explanation is structured into three parts: *Deep-Dive*, *Identified Problem*, and *Solution*.

Problem 1 - Predicted Link Type in Router LSA (Type 1 LSA)

Deep Dive into Link Types:

A Router LSA (Type 1 LSA) advertises the state of a router's own interfaces. Within this LSA, each interface is assigned a link type, which depends on the interface configuration and the observed topology on the LAN. The general rule for determining the link type is shown in Table 2.7:

Interface Configuration	Situation on the LAN	Link Type in Router LSA (Type 1 LSA)
passive[1]	Only one Router (nothing else is possible)	stub network
no special network definitions	only one Router	stub network
no special network definitions	two or more routers connected to the same LAN	transit network

Table 2.7: Link Types based on Interface State

[1] A passive interface is an interface that doesn't send or receive OSPF hello packets, but still participates in the routing process by learning routes from other OSPF routers on that interface.

Identified Problem:

The absence of a passive configuration on an interface does not necessarily indicate that it belongs to a transit network.

Solution:

We introduced a new network type called "unknown" for cases where an interface is expected to act as a transit network but cannot be confidently classified during prediction. If the shouldState marks a link as "unknown", and the actual isState turns out to be either a stub or transit network, this is treated as a valid match, no anomaly is raised in such cases.

Resulting Limitation:

Since the link type is yet another element in OSPF that can change dynamically at runtime, this solution introduces a limitation. It is technically possible for a non-passive interface to appear with a different link type than expected, without triggering an anomaly.

Problem 2 - Predicted Router LSA (Type 1 LSA) for P2P Interfaces

Deep Dive Point-to-Point Networks:

The behavior is clearly defined in RFC 2328, Section 12.4.1.2. The handling of interfaces in Router LSA (Type 1 LSA) generation depends on whether the interface is numbered or unnumbered:

Numbered[2]	Unnumbered[3]
Stub Network: If the neighboring router's IP address is known, the `Link ID` is set to that IP address. If the neighbor is not known, the subnet's IP address is used as the `Link ID`. **Point-to-Point:** The `Link ID` is set to the Router ID of the neighboring router.	**Point-to-Point:** Only a single link is advertised with link type point-to-point. The `Link ID` is set to the interface index (`ifIndex`) of the local router's interface.

Table 2.9: Difference of Router LSA (Type 1 LSA) from Numbered/Unnumbered Interfaces

[2] A numbered interface has an IP address and subnet assigned.
[3] An unnumbered interface has no IP address assigned; uses interface index as identifier.

Point-to-Point peerings in OSPF can be configured in several common ways:

- `/32` Address: The neighbor's address must be manually specified, as no subnet is implied.
- `/31` Network: The neighbor's IP address is automatically derived, as both usable addresses in the `/31` subnet are assumed to be point-to-point peers. This is supported due to RFC 3021 and is commonly used to conserve address space.
- `/30` Network: The neighbor's address is automatically derived from the subnet. The network and broadcast addresses are reserved, so only two usable host addresses remain.

It is also important to note that FRR treats interfaces configured with a `/32` address as unnumbered, which is documented in the official FRR documentation.

Identified Problem:
The industry partner's configuration deviates from RFC-specified standards. Despite this deviation, their LSDB is populated in a manner that appears functionally correct. The underlying interface operates as a peer-to-peer interface, as confirmed by examining the applied configuration using the command `ip ospf interface <interfaceName>`. This behavior occurs because the underlying interface is inherently peer-to-peer, and the Zebra daemon correctly identifies this configuration.

Solution:
To address this issue, the anomaly analyzer was adjusted to treat the interface as numbered, even though the official FRR documentation indicates that such configurations should be interpreted as unnumbered. This adjustment accurately reflects the standard behavior observed with the industry partner's setup.

Resulting Limitation:
A significant issue remains unresolved. The peer-to-peer interface parsing logic was specifically tailored to accommodate the industry partner's configuration. Consequently, `/32` point-to-point peering configurations will trigger false anomaly detection. This limitation will be addressed in future updates.

Anomaly Detection Limitations

Due to the decentralized nature of OSPF, the information available on a single host is limited to its local configuration scope. This constraint limits LSDB prediction capabilities to self-originated entries resulting from the router's own FRR configuration, such as interfaces and static routes. For example, self-originated AS External LSA (Type 5 LSA) entries redistributed from BGP cannot be predicted. Despite this limitation, it remains possible to reliably predict self-originating link states.

Once both the `isState` and `shouldState` have been parsed, the comparison process begins. At this point, the analysis encounters another challenge. Ideally, the anomaly detection should reliably determine whether a network functions as a stub or a transit network. In practice, however, this is not feasible with sufficient accuracy.

Chapter 3

Conclusion

This chapter evaluates the functional and non-functional requirements defined at the beginning of the project and concludes with an outlook on potential future developments of the FRR-MAD project beyond the scope of this thesis.

3.1 Outcome Analysis

This section presents a comprehensive evaluation of all defined Functional Requirements and Non-Functional Requirements from the project specification. Each requirement is individually assessed with regard to its implementation status, validation method, and critical discussion of outcomes and trade-offs. Optional features are also reviewed to determine their current level of completion and their potential for future development.

Legend for Evaluation Icons:

- ✅ MVP and Finished
- ❌ MVP and not Finished
- ✅ Optional and Finished
- ❌ Optional and not Finished

3.1.1 Functional Requirements Evaluation

MVP | **FR1-1:** Export OSPF Routing Metrics

Detail	The backend (frr-mad-analyzer) supports exporting relevant OSPF metrics in JSON format, which are directly polled from the router. A Prometheus-compatible endpoint has been implemented to access these metrics externally, enabling persistent storage of these metrics. The configuration file allows users to selectively define which metrics are exported. Due to the improved architecture, these export settings do not interfere with any functionality in the TUI, even if no metrics are exported. Additionally, the backend (frr-mad-analyzer) is designed to be easily extensible, making it straightforward to add support for additional metrics if needed.
Status	Fully implemented
Validation	Unit Tests, Acceptance Test with stakeholder
Discussion	The initial implementation of this feature was static in nature. However, to accommodate the industry partner's request to reduce the number of Prometheus labels, flexibility was incorporated into this feature. Consequently, individual metrics can be enabled or disabled through either the configuration file or command-line options. This feature includes enhancements beyond the original specification.

Optional | **FR1-2:** Export BGP Routing Metrics

Detail	This optional FR aimed to provide BGP metrics export functionality, similar to the OSPF implementation described in FR1-1 evaluation.
Status	Not implemented
Reason (if not completed)	Several challenges were encountered during the implementation of OSPF anomaly detection, as discussed in FR3 evaluation. Additionally, the scope of the MVP regarding both anomaly detection (FR3 evaluation) and information gathering (FR2 evaluation) was clearly underestimated.
Discussion	The decision to exclude BGP-related functionality was made in project week 12 and discussed with the stakeholder on May 8, 2025. This trade-off allowed the team to focus on completing and stabilizing OSPF-related features. Future work could include extending the system to support BGP metrics in a similar fashion to OSPF.

MVP | **FR2:** Gather FRR Routing Information

Detail	The backend (frr-mad-analyzer) includes a dedicated module for aggregating OSPF routing information from Free Range Routing (FRR). Using a fixed polling interval of 5 seconds, the system continuously collects data such as installed and advertised OSPF routes, various LSA types (1, 2, 3, 4, 5 and 7), and FRR system configuration and status. The aggregated data is made available for real-time analysis in the analyzer component and for visualization in the frontend (frr-mad-tui).
Status	Fully implemented
Validation	Unit Tests, Acceptance test with stakeholder
Discussion	Due to the lack of open-source tools for parsing FRR routing data, we developed a custom parser using FRR's JSON command output. This solution integrated well and met project needs. However, structural changes in FRR version 10.3 JSON responses broke compatibility. As a result, FRR-MAD currently supports only the stable 8.5 release. A future enhancement would be to extend parser compatibility across multiple FRR versions.

MVP | **FR3:** OSPF Route Anomaly Detection by TUI

Detail	The analyzer component implements OSPF route anomaly detection focusing on three key LSA types: Router LSA (Type 1 LSA), which check intra-area links and interfaces; AS External LSA (Type 5 LSA), which verify redistributed routes; and NSSA External LSA (Type 7 LSA), which validate NSSA-specific routes. Additionally, the system performs FIB-LSDB consistency checks to identify installed routes that don't match the LSDB. The detection covers unadvertised routes (missing expected routes) and overadvertised routes (unexpectedly announced routes).
Status	Partially implemented.
Validation	Unit tests for core detection logic and acceptance tests with stakeholder using real network scenarios
Reason (if not completed)	No examples were provided by the stakeholder to demonstrate how a Duplicated Route could occur. Additionally, no clear case was given for Wrongly Advertised Route that could not already be categorized as either Unadvertised Route or Overadvertised Route.
Discussion	The implementation provides comprehensive anomaly detection for the most critical LSA types affecting route propagation. The FIB-LSDB comparison adds valuable operational state validation. While Network (Type 2) and Summary (Type 3/4) LSAs aren't directly analyzed, their effects are captured through the router LSA and FIB checks. The modular design allows for future extension to additional LSA types if needed.
Comment	The system focuses on the LSA types that most directly impact route anomalies, providing targeted detection without unnecessary complexity

MVP | **FR4:** Query and Display Routing Anomalies via TUI

Detail	The frontend (frr-mad-tui) visualizes all required OSPF anomalies as defined in FR4, including affected prefixes and expected vs. actual values. Additionally, it covers further anomalies such as LSDB-to-RIB and RIB-to-FIB mismatches. The dedicated OSPF Monitor and RIB pages were developed to support troubleshooting. Anomaly data is queried automatically at regular intervals or upon accessing the OSPF dashboard.
Status	Partially implemented.
Validation	Acceptance test with stakeholder
Reason (if not completed)	Same reason as in FR3 evaluation.

Optional | **FR5:** BGP Route Anomaly Detection

Detail	-
Status	Not implemented
Comment	BGP-related functionality was entirely excluded from the project scope, as discussed in FR1-2 evaluation.

Optional | **FR6:** Adding new tabs to TUI

Detail	The config file enables control over which pages of the frontend (frr-mad-tui) are active. For instance, the shell page can be disabled while retaining full functionality of the remaining pages. On startup, the TUI reads these settings and renders only the activated pages.
Status	Fully implemented
Validation	Acceptance testing performed within the project team.
Discussion	The current implementation supports page-level toggling via the config file. A more granular approach allowing activation of individual sub tabs would improve modularity and even satisfy user needs.

Optional | **FR7:** OSPF Neighbor States Troubleshooting

Detail	The frontend (frr-mad-tui) offers a clear, focused display of OSPF neighbors, showing their states and adjacency details without clutter. Neighbor states are also included in the Router LSA (Type 1 LSA) Transit Network table, helping users determine if the local router, a direct neighbor, or another transit network device is the Designated Router (DR).
Status	Fully implemented
Validation	Acceptance test with stakeholder
Discussion	Enhancing the feature with state change detection and historical tracking would further improve troubleshooting capabilities.

Optional | **FR8:** BGP Session States Troubleshooting

Detail	-
Status	Not implemented
Comment	BGP-related functionality was entirely excluded from the project scope, as discussed in FR1-2 evaluation.

Optional | **FR9:** FR9: TUI History

Detail	All command inputs from the frontend (frr-mad-tui) client are logged to a dedicated log file. This applies to both the built-in vtysh shell and bash shell. Additional metadata including timestamps and source identifiers indicate when commands were executed and from which shell.
Status	Partially implemented.
Validation	Acceptance testing performed within the project team.
Reason (if not completed)	Not fully completed due to time constraints.
Discussion	The foundational infrastructure has been established. Interactions with frontend (frr-mad-tui) are already logged with sufficient metadata for proper categorization. Only an additional feature for input history lookup remains to be implemented.

Optional | **FR10:** Issue Solving via TUI

Detail	The Custom Shell enables users to execute vtysh commands directly within the TUI, allowing manual resolution of routing anomalies. This feature supports issue-solving functionality, but must be used with care due to the constraint defined in NFR10, which requires the router configuration to remain unchanged between frr-mad-tui startup and shutdown.
Status	Partially implemented
Validation	Acceptance testing performed within the project group.
Reason (if not completed)	Not fully completed due to time constraints.
Discussion	While the current shell feature enables manual issue resolution, a major improvement would be a preview mechanism that estimates the impact of a configuration change. Such a feature could simulate the resulting OSPF state based on current neighbors and received/sent LSAs. However, implementing this feature would require significant development effort and depends on other optional functionalities that exceed the time constraints of this thesis.

Optional | **FR11:** Guided Fixes for Misadvertised Routes

Detail	This feature was planned as an extension of the previous FR (FR10). For further details, see FR11.
Status	Not implemented
Reason (if not completed)	Not implemented due to time constraints and the unexpectedly large scope of work.
Discussion	This feature has the potential to provide significant support for the industry partner's mission control operator. The steps outlined in Industry Partner Specific Problems could potentially be executed with a single action.

Optional | **FR12:** Export Routing Anomaly Analysis Results

Detail	The frontend (frr-mad-tui) implements the feature "Export Options," accessible from every page that displays backend (frr-mad-analyzer) data. It allows users to export any parsed data, such as OSPF metrics, routing tables, or anomaly results directly to the clipboard or to a temporary file. The export path can be configured via the application's configuration file.
Status	Fully implemented
Validation	Acceptance testing with stakeholder
Discussion	The only reliable way to transmit a "copy to clipboard" command from a Docker container running on a VM (accessed via SSH)to the user's local terminal is through the OSC52 Protocol. For example, in iTerm2, this feature can be enabled under *Settings > General > Selection > Access: <allow>*. This feature includes enhancements beyond the original specification.

Optional | **FR13:** Knowledge Database for Manual Fixes

Detail	This feature was proposed spontaneously during a meeting but remains unimplemented due to its broad scope and lack of detailed planning.
Status	Not implemented
Reason (if not completed)	Not implemented due to time constraints and the unexpectedly large scope of work.
Discussion	Such a feature would help novice users by giving them access to insights from more experienced community members.

MVP | **FR14:** FRRMon Replacement

Detail	The system completely replaces FRRMon's export functionality while adding anomaly detection and interactive TUI capabilities. It maintains Prometheus compatibility while providing more comprehensive OSPF-specific checks than the original tool.
Status	Fully implemented
Validation	Successfully validated by the stakeholder through testing on their production systems, confirming all monitoring workflows work as expected.
Discussion	The solution successfully replaced FRRMon while offering superior capabilities. The main improvement is the interactive TUI interface for immediate troubleshooting. While more resource-intensive than FRRMon, the benefits of real-time analysis justify this trade-off. Future enhancements could include direct Prometheus export.

Optional | **FR15:** Dynamic Route Filtering in TUI

Detail	The frr-mad-tui provides dynamic filtering capabilities across all pages, allowing users to interactively filter routes or LSA contents during a session. The Filter is one of the core usability features, significantly improving and readability, especially in large-scale routing environments. Filters can be stored temporarily and re-applied across views, aligning with the design goals of FR15.
Status	Fully implemented
Validation	Acceptance testing with stakeholder
Discussion	This is already a highly practical feature within the FRR-MAD application. However, its utility could be further improved by extending the filter to support regular expression (regex) searches.

Optional | **FR16:** Export Detected Anomalies

Detail	The system fully implements anomaly export functionality to Prometheus, covering all detected OSPF anomalies including overadvertised and unadvertised routes. Metrics are categorized by source (Router, External, NSSA External, RIB-to-FIB, and LSDB-to-RIB) and include detailed contextual information about each anomaly. The exporter runs at a configurable interval with automatic retry logic for failed updates.
Status	Fully implemented
Validation	Unit Tests, Acceptance Test with stakeholder
Discussion	The implementation provides comprehensive anomaly reporting with detailed labels for effective monitoring. The modular design allows for easy extension to additional anomaly types. The retry mechanism ensures reliability during temporary failures. Future enhancements could include additional diagnostic metadata in the exported metrics.

3.1.2 Non-Functional Requirements Evaluation

NFR1: Presentation of information dashboard extension

Displays OSPF and system information in a dashboard view.

NFR2: Tab selection

Allows switching between TUI tabs via click or shortcut.

NFR3: Presentation of information BGP extension

Extends the dashboard to display BGP information.

NFR4: Correctness of frr_exporter metrics

Ensures the frr_exporter detects and exports anomalies within 120 seconds.

NFR5-1: TUI read mode only

Ensures the TUI operates in read-only mode by default.

NFR5-2: TUI read/write mode

Allows switching between read-only and read/write modes with visual cues.

NFR6: Limited exported metrics

Ensures the frr_exporter only exports OSPF- and BGP-related metrics.

NFR7: TUI User Experience

Ensures the TUI uses color schemes for warnings and informational messages.

NFR8: TUI Resolution Support

Ensures the TUI displays correctly at 1280×720 resolution.

NFR9: Lazy Loading of TUI

Ensures the TUI only loads content for the currently selected tab.

NFR10: Integration of TUI in Dev/Prod Environments

Ensures the TUI can be easily integrated into Linux environments.

NFR11: Guided or Automated Implementation of TUI

Provides a script to automate TUI installation in five commands.

NFR12: Testing Environment for frr_exporter and TUI

Ensures a test environment is provided for both frr_exporter and the TUI.

3.2 Future Directions

The FRR-MAD project successfully delivered its core OSPF monitoring and anomaly detection capabilities, but several areas remain for future development. Based on implementation challenges and stakeholder feedback, we identify four key improvement vectors:

3.2.1 Protocol Support Extensions

- **BGP integration**: Implement monitoring and anomaly detection for BGP (FR1-2, FR5, FR8) to create comprehensive routing visibility
- **FRR version compatibility**: Refactor the JSON parser to support newer FRR releases (post-8.5) as identified in FR2 evaluation

3.2.2 Anomaly Detection Enhancements

- **Guided remediation**: Develop the planned guided fixes system (FR11) with configuration impact simulation (FR10)
- **LSA coverage expansion**: Extend detection to Network (Type 2) and Summary (Type 3/4) LSAs (FR3 evaluation)

- **Interface generalization**: Refactor interface parsing to correctly handle `/32` point-to-point configurations and avoid false positives caused by current industry-specific adjustments (Problem 2)

3.2.3 User Experience Improvements

- **Intuitive shell tab**: Enhance the TUI shell with command autocompletion, history navigation, and syntax highlighting to improve interactive usage
- **Advanced filtering**: Implement regular expression support for dynamic filters (FR15)

3.2.4 Community and Operational Features

- **Knowledge base**: Implement the proposed troubleshooting database (FR13)
- **Enhanced metrics**: Expand Prometheus exports with diagnostic metadata (FR16)

This roadmap prioritizes features that would most significantly enhance operational utility while maintaining the system's modular architecture. The proposed extensions address both immediate technical gaps and long-term usability requirements identified during the project.

Part III

Project Documentation

Chapter 4

Requirements

This chapter defines the system requirements, starting with Personas and Actors to identify key users and interactions. A Use Case Diagram provides an overview, followed by detailed Functional Requirements for system capabilities. Finally, Non-Functional Requirements ensure the system meets standards for capability, reusability, security, aesthetic, serviceability and testability forming a solid foundation for development.

4.1 Personas

Two personas with distinct and representative characteristics have been created to reflect typical users of the application. These personas serve as a tool to better define and refine the application's functional and non-functional requirements.
The first persona represents a fictional employee at the company Open Systems. "Ronny Router" is the primary persona for whom the application is designed. The second persona represents anyone else interested in improving the monitoring of their FRRouting OSPF setup.

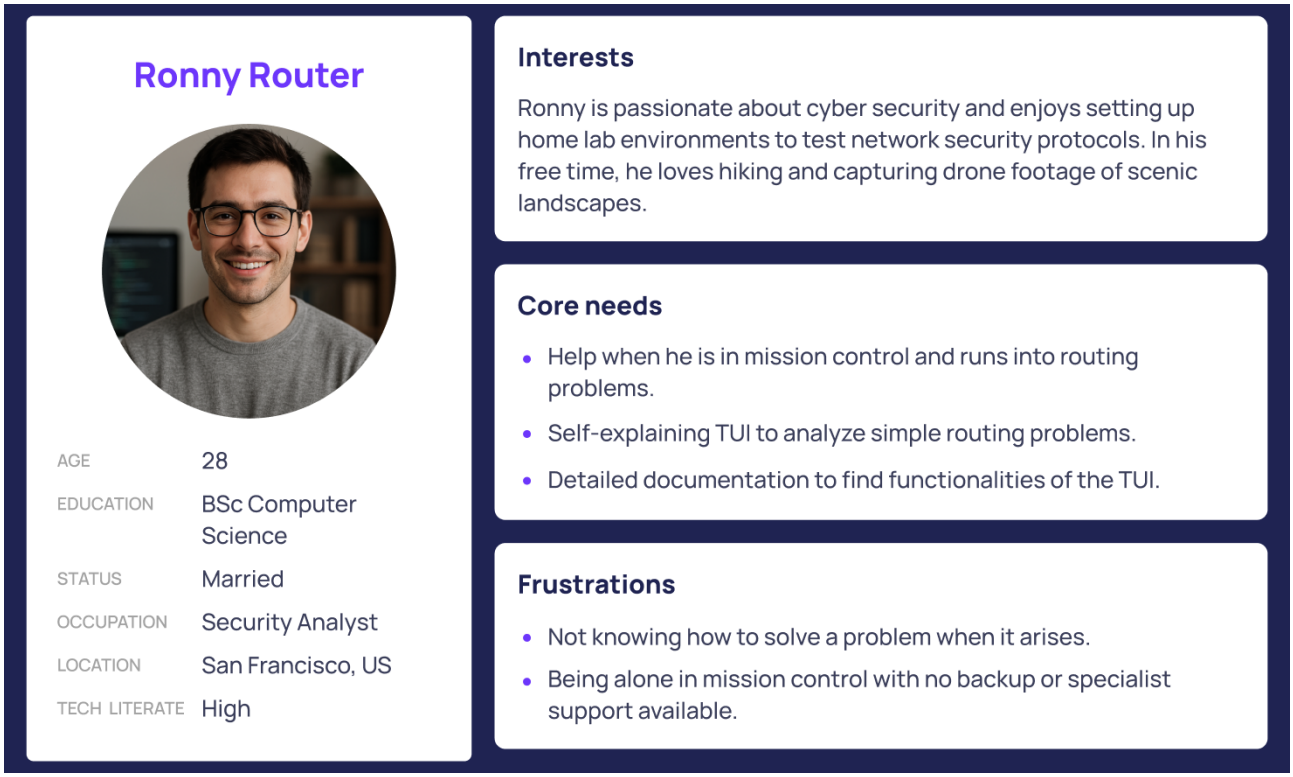


Figure 4.1: Persona - Mission Control Operator

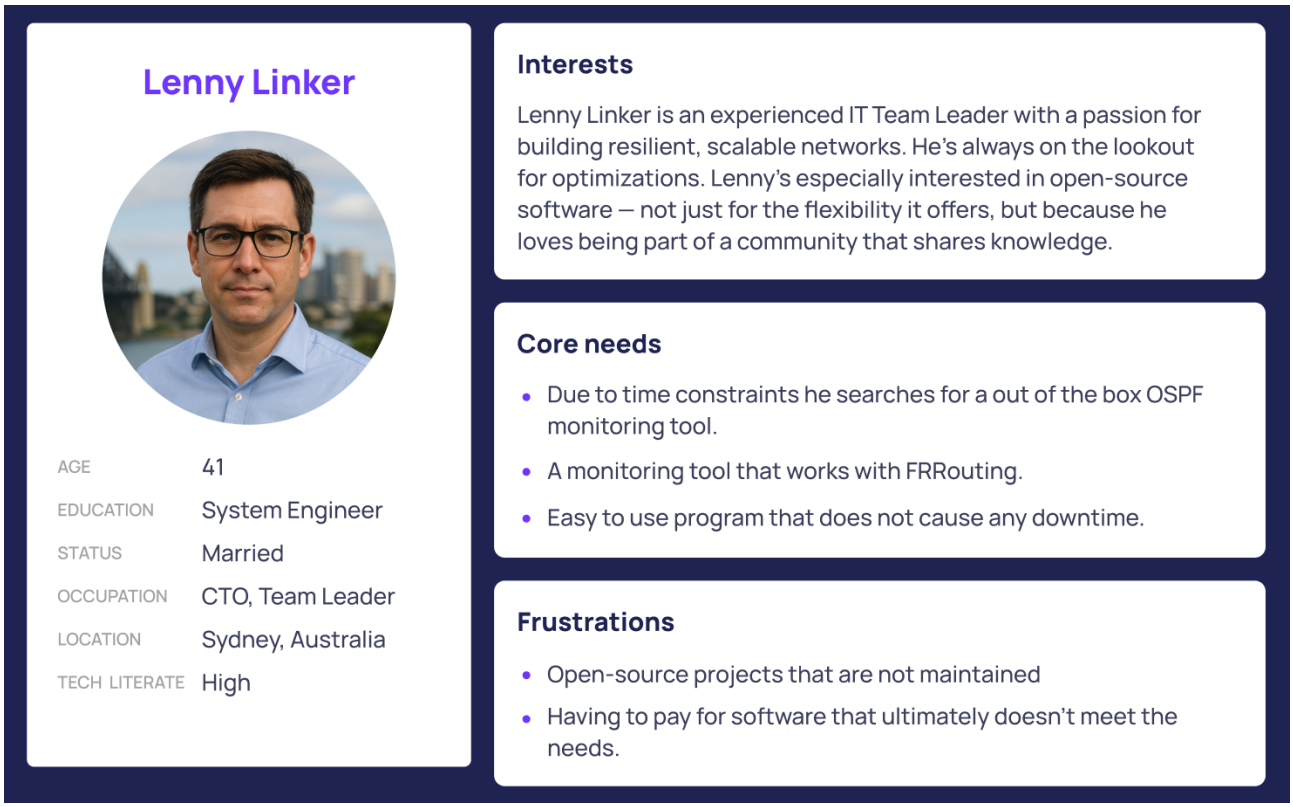


Figure 4.2: Persona - anyone else using our application

The persona cards were designed using Figma, and the profile pictures generated with ChatGPT[21],[10].

4.2 Actors

- **Mission Control Operator**: An employee who uses frr-mad-tui to see and solve network anomalies.
- **System**: The system collects data and is looking for network anomalies.

4.3 Use Case Diagram

This section shows the use case diagram, which illustrates the functional requirements from the perspective of the actors.

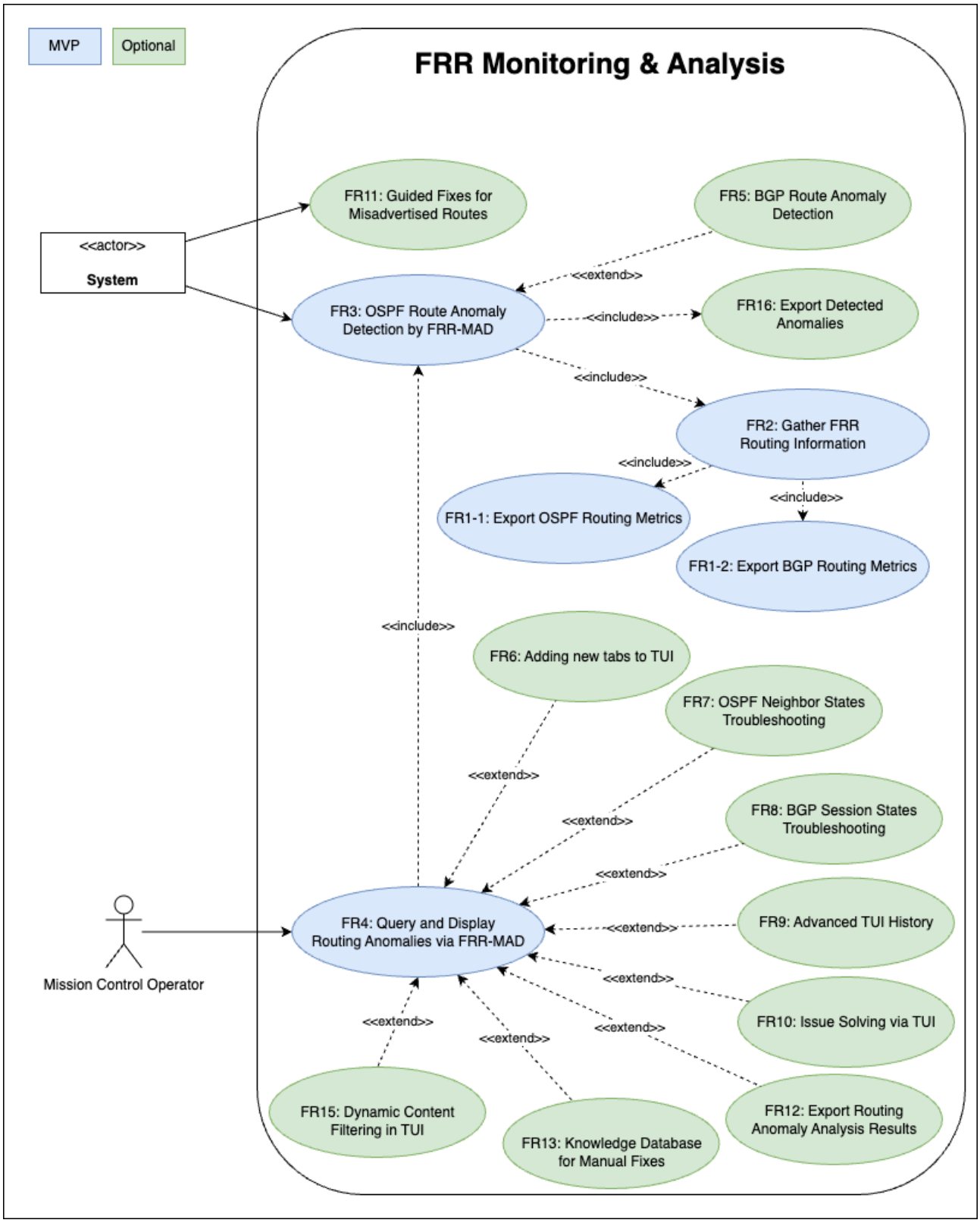


Figure 4.3: Use Case Diagram

4.4 Functional Requirements

This section details the Functional Requirements of the system, with a focus on the Minimal Viable Product (MVP). Each requirement ensures the implementation of essential Open Shortest Path First (OSPF) monitoring and analysis functionalities for Free Range Routing (FRR). In addition to the MVP, the system also includes further FRs, such as Border Gateway Protocol (BGP) monitoring and other advanced features, which are planned for implementation as part of the complete system.

FR	Description
FR1-1: Export OSPF Routing Metrics	Exports OSPF routing metrics to Prometheus.
FR1-2: Export BGP Routing Metrics (Optional)	Exports BGP advertisement metrics to Prometheus.
FR2: Gather FRR Routing Information	Collects OSPF and FRR routing data for real-time analysis.
FR3: OSPF Route Anomaly Detection by FRR-MAD	Detects OSPF-specific anomalies like unadvertised or overadvertised routes.
FR4: Query and Display Routing Anomalies via FRR-MAD	Allows users to query and display routing anomalies in the TUI.
FR5: BGP Route Anomaly Detection (Optional)	Detects BGP-specific anomalies like unadvertised or duplicated routes.
FR6: Adding new tabs to TUI (Optional)	Customizability for the different tabs.
FR7: OSPF Neighbor States Troubleshooting (Optional)	Displays detailed OSPF neighbor states and adjacency information.
FR8: BGP Session States Troubleshooting (Optional)	Displays detailed BGP session states and peer information.
FR9: Advanced TUI History (Optional)	Saves a log file and provides a historical lookup for past commands.
FR10: Issue Solving via TUI (Optional)	Allows users to manually apply fixes for routing anomalies.
FR11: Guided Fixes for Misadvertised Routes (Optional)	Suggests and automatically applies fixes for detected anomalies.
FR12: Export Routing Anomaly Analysis Results (Optional)	Exports anomaly analysis results to files (e.g., CSV, JSON).
FR13: Knowledge Database for Manual Fixes (Optional)	Saves manual fixes to a Knowledge Database for future reference.
FR14: FRRMon Replacement	Provides equivalent anomaly detection capabilities as FRRMon with added TUI interface.
FR15: Dynamic Route Filtering in TUI (Optional)	Provides temporary filtering of displayed routes through interactive TUI controls.
FR16: Export Detected Anomalies (Optional)	Exports the detected anomalies to Prometheus.

Table 4.1: Summary of Functional Requirements

4.4.1 FR1-1: Export OSPF Routing Metrics

Main Success Scenario (MVP)

- The system provides an exporter to expose **OSPF** Routing metrics via a Prometheus-compatible endpoint.
- The exported OSPF data includes:
  - Number of installed and advertised OSPF routes.
  - Differentiated OSPF traffic statistics.
  - OSPF LSA Types and counts.
- The data is formatted for seamless integration with existing monitoring systems.

Alternate Scenarios

- If exporting fails, the system retries the process and logs the failure.

4.4.2 FR1-2: Export BGP Routing Metrics (Optional)

Main Success Scenario (Optional Feature)

- The system provides an exporter to expose **BGP** Routing metrics via a Prometheus-compatible endpoint.
- The exported BGP data includes:
  - Number of installed and advertised BGP routes.
  - Differentiated BGP traffic statistics.
- The data is formatted for seamless integration with existing monitoring systems.

Alternate Scenarios

- If exporting fails, the system retries the process and logs the failure.

4.4.3 FR2: Gather FRR Routing Information

Main Success Scenario (MVP)

- The system continuously retrieves routing information from FRR for real-time analysis.
- The collected data includes:
  - Installed and advertised OSPF routes.
  - OSPF-specific details such as **Link State Advertisement (LSA)** **Type 1, 2, 3, and 5**.

  - FRR System configuration and status.

Optional Features (Future Enhancements)

- Collection of **BGP routes** And protocol-specific attributes.
- Additional OSPF data such as **detailed adjacency states**.

Alternate Scenarios

- If FRR does not provide sufficient data, the system logs an error.

4.4.4 FR3: OSPF Route Anomaly Detection by FRR-MAD

Main Success Scenario (MVP)

- Upon entering the TUI it detects **OSPF-specific anomalies** Specifically looking at inconsistencies:
  - **Unadvertised Route** – Expected OSPF routes that are missing.
  - **Overadvertised Route** – Routes that should not be advertised but are still announced.
  - **Wrongly Advertised Route** – Incorrectly advertised prefixes (e.g., '10.39.0.0/27' instead of '10.39.0.0/17').
  - **Duplicated Route** – The same OSPF route is advertised multiple times with different attributes (e.g., different costs or from multiple LSAs).

Alternate Scenarios

- If no anomalies are detected, the system confirms OSPF route consistency.
- If an anomaly is detected but cannot be classified, the system logs it for manual review.

4.4.5 FR4: Query and Display Routing Anomalies via FRR-MAD

Main Success Scenario (MVP)

- The TUI allows users to query and display the following routing anomalies:
  - **Unadvertised Route**: Expected routes that are missing.
  - **Overadvertised Route**: Routes that should not be advertised but are still announced.
  - **Wrongly Advertised Route**: Incorrectly advertised prefixes (e.g., '10.39.0.0/27' instead of '10.39.0.0/17').
  - **Duplicated Route**: The same route advertised multiple times with conflicting attributes.
- The tool provides a clear and concise summary of detected anomalies, including:

  - Affected prefixes.
  - Expected vs. actual route attributes.
  - Protocol (OSPF).

Alternate Scenarios

- If no anomalies are detected, the tool confirms that the routing information is consistent.
- If data is missing or incomplete, the tool logs an error and suggests checking the prometheus database or FRR service.

4.4.6 FR5: BGP Route Anomaly Detection (Optional)

Main Success Scenario (Optional Feature)

- The system detects **BGP-specific anomalies** Specifically looking at inconsistencies:
  - **Unadvertised Route** – Expected BGP routes that are missing.
  - **Overadvertised Route** – Routes that should not be advertised but are still announced.
  - **Wrongly Advertised Route** – Incorrectly advertised prefixes (e.g., '10.39.0.0/27' instead of '10.39.0.0/17').
  - **Duplicated Route** – The same route being advertised multiple times with conflicting attributes.

Alternate Scenarios

- If BGP route inconsistencies are detected, the system alerts the user.
- If no anomalies are detected, the system confirms BGP route consistency.

4.4.7 FR6: Adding new tabs to TUI (Optional)

Main Success Scenario (Optional Feature)

- The TUI initially will provide basic functions for information gathering and troubleshooting.
- These Information can be accessed via tabs. The base tab is a dashboard, which will contain information for troubleshooting purposes.
- On startup, the system reads the config file's list of pages and their activated flags.
- The TUI renders only those pages marked as activated, with all navigation keys intact.

Alternate Scenarios

- Base tabs are immutable.

- If all pages are deactivated, the TUI displays only the Dashboard (which always remains active).
- Any entries that aren't valid TUI pages are ignored.

4.4.8 FR7: OSPF Neighbor States Troubleshooting (Optional)

Main Success Scenario (Optional Feature)

- The TUI provides detailed information about OSPF neighbor states and adjacency status this is an extension from FR4.
- The tool displays:
  - Current OSPF neighbor states (e.g., Full, 2-Way, Down).
  - Adjacency details (e.g., neighbor IP, interface, state).
- The tool allows users to filter and search for specific OSPF neighbors or interfaces.

Alternate Scenarios

- If OSPF neighbor data is unavailable, the tool logs an error and suggests checking the prometheus database or FRR configuration.

4.4.9 FR8: BGP Session States Troubleshooting (Optional)

Main Success Scenario (Optional Feature)

- The TUI provides detailed information about BGP session states and peer status this is an extension from FR5.
- The tool displays:
  - Current BGP session states (e.g., Established, Idle, Active).
  - Peer details (e.g., peer IP, AS number, uptime).
  - Timestamps for session state changes.
- The tool allows users to filter and search for specific BGP peers or sessions.

Alternate Scenarios

- If BGP session data is unavailable, the tool logs an error and suggests checking the prometheus database or FRR configuration.

4.4.10 FR9: Advanced TUI History (Optional)

Main Success Scenario (Optional Feature)

- A log file is saved on the device.
- The TUI provides a historical lookup feature(configurable).
- The historical lookup is compromised of three parts.
  - Issue History (e.g., misadvertised route)
  - Command used History (e.g., show ip bgp neighbors)
  - A combination of both items above.

Alternate Scenarios

- If the history is empty, show a notice, that no history is available.

4.4.11 FR10: Issue Solving via TUI (Optional)

Main Success Scenario (Optional Feature)

- The TUI allows users to manually apply fixes for detected routing anomalies.
  - The TUI provides some kind of compatibility layer to execute vtysh commands.
- The TUI provides a before and after changes result to preview changes before applying them.

Alternate Scenarios

- If a fix cannot be applied (e.g., due to insufficient permissions), the tool logs an error and suggests manual intervention.

4.4.12 FR11: Guided Fixes for Misadvertised Routes (Optional)

Main Success Scenario (Optional feature)

- This is an enhancement to fr10, building upon its functionality.
- If the system detects a misadvertised or missing route, it suggests possible corrective actions.
- The system provides a **recommended command** To resolve the issue, such as updating an access list or modifying a route advertisement.
- If enabled by the user, the system **automatically applies the fix** (e.g., executing a corrective command in frr) after prompting for confirmation.

Alternate Scenarios

- If an issue is detected but no fix is confidently determined, the system logs the anomaly and suggests manual intervention.

4.4.13 FR12: Export Routing Anomaly Analysis Results (Optional)

Main Success Scenario (Optional feature)

- The TUI allows users to export the results of the routing anomaly analysis to a file.
- The exported data includes:
  - Detected anomalies (over-advertised, unadvertised, wrongly advertised, duplicated).
  - Timestamps for when the analysis was performed.

Alternate Scenarios

- If the export fails, the tool logs an error and retries the process.

4.4.14 FR13: Knowledge Database for Manual Fixes

Main Success Scenario (Optional feature)

- The system provides a Knowledge Database where manual fixes applied by users are saved for future reference.
- when a user applies a manual fix (e.g., modifying access lists, re-advertising routes), the system prompts the user to save the fix to the Knowledge Database.
- The saved fix includes:
  - A description of the issue (e.g., “unadvertised route for prefix 10.39.0.0/27”).
  - The steps taken to resolve the issue (e.g., “modified access list to allow route advertisement”).
  - The protocol affected (e.g., OSPF, BGP).
  - The timestamp of when the fix was applied.
- The Knowledge Database is searchable by keywords, protocol, or issue type.
- When a similar issue is detected, the system suggests relevant fixes from the Knowledge Database to the user.
- The format of the Knowledge Database is portable and deployable.

Optional Features (Future Enhancements)

- User ratings: Allow users to rate the effectiveness of fixes in the Knowledge Database.
- Automated fix suggestions: automatically apply fixes from the Knowledge Database if the issue matches a known problem.
- Integration with external knowledge bases: allow the system to pull fixes from external knowledge bases or forums.

Alternate Scenarios

- If the user chooses not to save a fix, the system logs the fix locally but does not add it to the Knowledge Database.

4.4.15 FR14: FRRMon Replacement

Main Success Scenario (MVP)

- The system provides the anomaly detection capabilities currently available in FRRMon.
- The TUI interface adds new functionality for interactive querying and display of routing anomalies.
- Existing monitoring workflows using FRRMon can be fully replaced by this solution.

Alternate Scenarios

- If any FRRMon functionality is missing, the system logs an error.

4.4.16 FR15: Dynamic Content Filtering in TUI (Optional)

Main Success Scenario (Optional Feature)

- The TUI provides interactive controls for temporary filtering of anomaly and monitoring content.
- Users can quickly toggle visibility of specific subnet sizes during a session.
- Dynamic filters don't persist after TUI restart unless explicitly saved to configuration.
- The filter can be saved even when deactivated, allowing to easily apply it on another page.

Alternate Scenarios

- When no dynamic filters are active, the view follows static configuration settings.
- Filter changes are immediately reflected in the current view without requiring restart.

4.4.17 FR16: Export Detected Anomalies (Optional)

Main Success Scenario (Optional Feature)

- The system exports anomaly metrics to Prometheus at a configurable interval.
- The following anomaly types are exported:
  - Overadvertised routes
  - Unadvertised routes

- Anomalies are categorized by source:
  - RouterAnomaly
  - ExternalAnomaly
  - NssaExternalAnomaly
  - RibToFib
  - LsdbToRib
- Detailed anomaly information includes:
  - Interface address
  - Link state ID
  - Prefix length
  - Link type
  - P-bit status
  - Options

Alternate Scenarios

- When no anomalies are detected, all metrics are set to zero.
- If anomaly data is unavailable, the export is skipped.
- Failed updates are retried once before being logged as errors.

4.5 Non-Functional Requirements

Non-Functional Requirements (NFRs) are an integral part to proper project and the development of lean and functional applications. Thus, we'll be using the model for classifying software quality attributes FURPS+[12].

NFR	Description
NFR1: Presentation of Information Dashboard Extension	Displays OSPF and system information in a dashboard view.
NFR2: Tab Selection	Allows switching between TUI tabs via click or shortcut.
NFR3: Presentation of Information BGP Extension	Extends the dashboard to display BGP information.
NFR4: Correctness of frr_exporter Metrics	Ensures the frr_exporter detects and exports anomalies within 120 seconds.
NFR5-1: TUI Read Mode Only	Ensures the TUI operates in read-only mode by default.
NFR5-2: TUI Read/Write Mode	Allows switching between read-only and read/write modes with visual cues.
NFR6: Limited Exported Metrics	Ensures the frr_exporter only exports OSPF and BGP-related metrics.
NFR7: TUI User Experience	Ensures the TUI uses color schemes for warnings and information.
NFR8: TUI Resolution Support	Ensures the TUI displays information correctly at 1280x720 resolution.
NFR9: Lazy Loading of TUI	Ensures the TUI only loads content for the currently selected tab.
NFR10: Integration of TUI in Dev/Prod Environments	Ensures the TUI can be easily integrated into Linux environments.
NFR11: Guided or Automated Implementation of TUI	Provides a script to automate TUI installation in 5 commands.
NFR12: Testing Environment for frr_exporter and TUI	Provides a containerlab-based testing environment for OSPF and BGP.

Table 4.2: Summary of Non-Functional Requirements

4.5.1 NFR1: Presentation of Information Dashboard Extension

Target Application: TUI
Category: Functionality → Capability
Description: As a mission control operator, I want to display information described in fr3 in one dashboard like view.
Acceptance Criteria: The dashboard is not bloated but still allows the mission control operator to get an overview of the system, running service and current issues.
Verification Process:

- On systems with and without issues open the TUI, the dashboard is the greeting window.
- Base information as described in the description should be visible.
- Regular review with stakeholder.

Verification Period: At the end of sprint 6.

4.5.2 NFR2: Tab Selection

Target Application: TUI
Category: Functionality → Capability
Description: As a mission control operator, I want to switch between the different tabs, for example go from dashboard tab to OSPF tab, by either clicking on the tab or using a shortcut command.
Acceptance Criteria: Switching between tabs works by click or shortcut.
Verification Process:

- Open TUI and switch between tabs

Verification Period: At the end of sprint 3.

4.5.3 NFR3: Presentation of Information BGP Extension

Target Application: TUI
Category: Functionality → Reusability
Description: As a mission control operator, I want to display BGP information such as described in the dashboard view, effectively extending NFR1. all the information should fully fit in 1280x1440 resolution.
Acceptance Criteria: The dashboard is extended by BGP information. Scrolling works if resolution doesn't fit description.
Verification Process:

- The dashboard always contains BGP information, if they are configured.
- Regular review with stakeholder.

Verification Period: At the end of sprint 8.

4.5.4 NFR4: Correctness of frr_exporter Metrics

application: FRR Exporter
Category: Functionality → Capabilities
Description: As a mission control operator, I want the frr_exporter To export anomalous behavior, as described in fr1 At the latest after two polling, with a 60 second polling interval.
Acceptance Criteria: At the latest after 120 seconds after a change in the routing occurs, the frr_exporter should display these changes after two polling.
Verification Process:

- Manual review of exported information.
- Automated comparison of information for the same time frame and FRR Exporter target.
- Automated comparison of advertised routes to a specific router.

Verification Period: At the end of each monitoring development sprint.

4.5.5 NFR5-1: TUI Read Mode Only

application: TUI
Category: Functionality → Security
Description: As a mission control operator, I want the TUI to have a read/write mode. Switching is done by either clicking on a switch or using a shortcut command. There is a visual cue depending on which mode I am in.
Acceptance Criteria: A toggle enables switching between read and read/write mode. The TUI should be displayed differently so that read/write mode can be perceived and toggling should be possible by clicking on a switch or using a shortcut command.
Verification Process:

- Click or execute the shortcut command - swapping between read/write mode.
- Switching between read and write mode should be visible by obvious TUI coloration changes.
- Config alteration of frr settings should be possible.

Verification Period: At the end of sprint 6.

4.5.6 NFR5-2: TUI Read/Write Mode

application: TUI
Category: Functionality → Security
Description: As a mission control operator, I want the TUI to be primarily read only to prevent accidental configuration changes. No changes can be applied in read only mode.
Acceptance Criteria: The TUI should display all necessary information for debugging purposes but

not allow any actions that could change the running configuration.
Verification Process:

- Take steps to not allow some kind of taint style vulnerabilities, considering that the TUI should be extensible. - adding some kind of white or black list filtering.
- Trying to execute taint style commands in different fashions. (should be automated)

Verification Period: Security tests in every sprint, with detailed audit log reviews at the end of each release cycle.

4.5.7 NFR6: Limited Exported Metrics

Target Application: FRR Exporter
Category: Functionality → Security (Information Disclosure)
Description: As a mission control operator, I want the frr_exporter to only export OSPF and BGP related information as described in fr2.
Acceptance Criteria: Only the necessary information, such as service status, announced routes, and similar should be displayed.
Verification Process:

- Auditing of monitoring output.
- The exporter only exposes information that is truly necessary for monitoring of OSPF and BGP.

Verification Period: At the end of each monitoring development sprint.

4.5.8 NFR7: TUI User Experience

application: TUI
Category: Usability → Aesthetics
Description: As a mission control operator I want the TUI to display warnings, configured information and running information with different colors.
Acceptance Criteria: As described the different information groups will follow a coloring scheme. A small legend will explain the scheme.
Verification Process:

- Open TUI, browse all tabs.
- Information coloration matches according to description.

Verification Period: At the end of sprint 6.

4.5.9 NFR8: TUI Resolution Support

application: TUI
Category: Usability → Aesthetics
Description: As a mission control operator, I want the TUI to display information, such as warnings and OSPF elements in groups on a resolution of at least 157 columns (width) and 38 lines (height).
Acceptance Criteria: On a terminal with resolution of 157 columns by 38 lines, the related information are properly displayed in their groups.
Verification Process:

- Frames separate different information objects.
- Scrolling should be kept to a minimum.
- Resolution should be at least 157 columns by 38 lines.

Verification Period: At the end of sprint 4.

4.5.10 NFR9: Lazy Loading of TUI

Category: Performance → Resource Consumption
Description: As an mission control operator, I want the TUI to only load currently selected tab contents. No other tab-specific content should be run otherwise.
Acceptance Criteria: Starting the TUI only the selected tabs will execute actions interactive with the system and processes reduction of system resource consumption.
Verification Process:

- Start TUI.
- Follow log output which commands are run.
- Switch tab and verify if tabs are lazy loaded.

Verification Period: At the end of project.

4.5.11 NFR10: Integration of TUI in Dev/Prod Environments

Target Application: TUI
Category: Supportability → Serviceability
Description: As a gnu/Linux custom distro maintainer, I want to integrate the TUI into my environment. Adding the package shouldn't be harder than copy the file to the system, make it executable and run it.
Acceptance Criteria: An installation process or guide should be present.
Verification Process:

- Review directives on where to place the binary, config file and DSL files.

- Manually review if the TUI is working as intended.

Verification Period: At the end of each TUI development sprint.

4.5.12 NFR11: Guided or Automated Implementation of TUI

Target Application: TUI
Category: Supportability → Reproducible Installation
Description: As a gnu/Linux custom distro maintainer, I want to be able to automate the process as described in NFR7. The TUI should be up and running within 5 commands.
Acceptance Criteria: As described the application needs to be functional after at most 5 commands have been executed. Acceptable results are only such if the application is fully functional.
Verification Process:

- Download the repository as a git repo or tarball
- Regardless either git has to be installed or the tarball needs to be extracted
- Execute the commands as described in the readme
- The TUI needs to be up and running with all base functions working.

Verification Period: Recurring thus at the end of every TUI development sprint.

4.5.13 NFR12: Testing Environment for frr_exporter and TUI

Category: Supportability → Testability
Description: As a frr user, I want to be able to deploy the test environment using containerlab[9]. Executing the run.sh script in the containerlab folder should start the environment.
Acceptance Criteria: Executing the run.sh script inside the containerlab folder starts up the whole containerlab and adds custom interface settings to the various PC nodes.
Verification Process:

- The testing environment can be deployed as described.
- The monitoring solution can be observed as described.
- OSPF is are running as described in the clab file.

Verification Period: At the end of development cycle for the MVP.

Chapter 5

Domain Analysis

To make sure our project doesn't already exist, we mainly look at related work in this chapter. The Open Shortest Path First (OSPF) and Border Gateway Protocol (BGP) protocols are the primary focus.

5.1 Related Work

Our goal is to evaluate the different available solutions, compare them against our requirements and find overlapping elements.

5.1.1 FRR Exporter

The *frr_exporter* project [29] is an open-source Prometheus exporter designed to collect metrics from Free Range Routing (FRR). It provides basic statistics such as the number of routes, protocol states, and up time per routing protocol instance.

While frr_exporter already covers the functional requirement FR2, it lacks more advanced capabilities such as:

- OSPF information such as
  - Advertised OSPF routes,
  - Installed OSPF routes and,
  - LSA types and counts (1,2,3,5).
- Running FRR config on the current instance. This includes OSPF and BGP.

For these reasons, we consider frr_exporter to be a suitable base for extension but insufficient as a standalone solution.

5.1.2 NetworkMonitor

The *NetworkMonitor* project [3] is another open-source monitoring tool designed for general network monitoring, with a focus on OSPF and BGP. It provides visualizations and some historical data tracking but does not offer Free Range Routing-specific insights or anomaly detection tailored to FRR's operational behavior.

5.1.3 Quagga Exporter

Historically, monitoring solutions existed for Quagga, FRR's predecessor. The project *quagga_exporter*, originally maintained at `https://github.com/teran/Quagga_exporter`, is no longer available. Its successor, *google-quagga* [1], primarily focuses on integrating Quagga with Google's internal monitoring ecosystem. This approach, however, is tightly coupled to Google's infrastructure and lacks the general-purpose flexibility required for our environment. Furthermore, it does not address anomaly detection, route alerting, or container-based deployment, making it unsuitable for direct reuse.

5.1.4 Batfish

The *Batfish* project [23] is a widely used open-source tool for static analysis of network configurations, supporting major protocols such as OSPF and BGP. It allows operators to validate configurations by detecting misadvertised or missing routes, identifying policy inconsistencies, and simulating various network events. Although Batfish excels at offline analysis, it does not provide real-time monitoring of running environments. Consequently, Batfish serves as inspiration for anomaly detection techniques, but it does not provide a directly applicable solution for our use case.

5.1.5 GoBGP CLI

The *GoBGP* project [22] is an open-source BGP implementation providing a comprehensive command-line interface (CLI) for analyzing BGP state and debugging route announcements. This tool demonstrates how to build powerful route inspection capabilities within a CLI, which directly conforms to the design of our own CLI-based troubleshooting tool. However, GoBGP is BGP-specific and does not cover OSPF or FRR-specific functionality, meaning it does not meet all our requirements.

5.1.6 OpenBMP

The *OpenBMP* project [25] is a real-time BGP Monitoring Protocol (BMP) collector. It aggregates BGP updates and state changes from routers and provides structured data suitable for integration with monitoring system. While OpenBMP excels at collecting BGP data, it lacks direct support for OSPF, real-time anomaly detection. It also requires BMP support on routers, which is not universally available in all FRR environments.

5.1.7 OSPF Topology Watcher

The *OSPF Topology Watcher* [30] monitors OSPF Link State Advertisements (LSAs) and builds a live view of the OSPF topology. It can detect missing neighbors, unexpected topology changes, and possible routing inconsistencies. While this covers one of our key requirements, detection of routing anomalies, it's OSPF-only and lacks Prometheus compatibility. Nevertheless, its anomaly detection approach provides useful techniques we can adopt.

5.1.8 Proprietary Network Monitoring Tools

Numerous commercial solutions provide comprehensive network monitoring, including tools such as:

- SolarWinds Network Performance Monitor
- PRTG Network Monitor
- ManageEngine OpManager

These tools already offer capabilities such as real-time route state visualization, historical data analysis, and alerting based on routing anomalies. However, all these solutions are closed-source and expensive. Because of its closed-source nature, extensibility of this tool is limited. Consequently, these tools serve as evidence that the need for an advanced, open source FRR monitoring and troubleshooting solution is needed.

5.1.9 Nvidia Cumulus Linux Documentation

Nvidia's Cumulus Linux documentation [18] provides valuable insight into how Free Range Routing is monitored and managed in production-grade environments. This includes recommended systemd integration, essential commands for health checks, and general operational guidelines. While this is not a standalone tool, it directly influences our understanding of what health metrics are considered critical in real-world FRR deployments.

5.1.10 Summary

After a thorough evaluation of the available tools, we concluded that none fully met our requirements for real-time FRR monitoring and anomaly detection. While *frr_exporter* provided valuable insights, such as Prometheus integration and command execution, it did not suite fully our need and needed an heavy revision.

Notably, *frr_exporter* served as a key reference for its lightweight design and metrics collection approach, directly inspiring parts of our implementation. However, we opted to develop a custom solution to address gaps in advanced route analytics, multi-protocol support, and CLI-driven debugging. Our tool synthesizes lessons from these projects while introducing novel features tailored to FRR's operational needs.

5.1.11 Decisions on Tooling Implementation

After evaluating the available tools and solutions for our monitoring system, we have made the following key decisions:

GoBGP Exclusion

While GoBGP provides a robust BGP implementation and CLI tooling, we have decided not to use it in our solution for several reasons:

- **Primary Use Case Mismatch**: GoBGP is primarily designed to run as a BGP server rather than as a monitoring tool for existing FRR implementations.
- **Protocol Focus**: Our project requires comprehensive OSPF monitoring with BGP being an optional component, while GoBGP focuses exclusively on BGP.
- **Integration Complexity**: Implementing GoBGP alongside FRR would introduce unnecessary complexity for our specific testing needs.

Given that BGP monitoring is optional in our project and not part of the MVP requirements, we will focus our development efforts on OSPF monitoring first, keeping the architecture flexible for potential BGP support in the future.

Batfish Implementation Strategy

Our evaluation of Batfish yielded the following conclusions:

- **Post-MVP Feature**: While Batfish provides excellent static configuration analysis capabilities, we have determined it falls outside our MVP scope of live anomaly detection.
- **Future Integration**: We will keep Batfish in consideration for later implementation phases as it offers valuable out-of-the-box configuration analysis features.
- **Architecture Considerations**: The system design will maintain compatibility with Batfish's requirements to facilitate easier integration when prioritized.

MVP Focus

The MVP implementation will concentrate on:

- Live anomaly detection in OSPF operations
- Basic BGP monitoring capabilities (as stretch goals)
- CLI-based troubleshooting

This focused approach ensures we deliver core functionality while maintaining a clear path for future enhancements including Batfish integration and BGP support.

Chapter 6

Solution Strategy

This chapter talks about the reasoning which technologies made for this project. Usually architecture and quality measures are also part of solution strategy. But because of their importance, they will be detailed in dedicated chapters.

6.1 Technology & Technique

The technologies employed in this bachelor's thesis are fairly standard. Because of the complexity of this project, it was decided to stress the importance of proper technology decisions. Therefore this section talks about the different environments, technologies and techniques employed. First, a brief overview will be presented of what the next section discusses.

- Development environment and all it's tool necessary to create it.
- Automated development steps to help test and build during development.
- Organizational and further miscellaneous decisions, to streamline communication and development.

6.1.1 Development

The development environment talks about which languages, frameworks, libraries and tools are used.

Language	Usage	Reasoning
Go	Main language to develop the project results.	In accordance with the project bachelor's thesis assignment go was the first choice. Additionally each member already possessed considerate expertise with go.
Shell	Create different helper function for deployment and building purposes.	The target system of the application was meant to be a unixGNU Linux system.
LaTeX	Utilized to write this bachelor's thesis paper.	Well-known and widely used in the science community. Wide support and many examples available.

Table 6.1: Programming Languages & Scripts

Language	Usage	Reasoning
bubbletea	Used to create a complex Text-Based User Interface (TUI).	A Framework[4] providing all necessary tools and libraries to create an interactive TUI. It has many available examples and a very good documentation.
lipglos	Used to create a more complex TUI.	A companion[5] to bubbletea, extending its capabilities. It only makes sense to use lipgloss to increase the capabilities of the TUI
protobuf	Both components, frr-mad-tui and frr-mad-analyzer require similar datatypes, thus a serializing structure was chosen.	Protocolbuffer[11] is easy to use and easy to implement. There are available proto compilers to create go compatible syntax. It also creates all the necessary datatype handlers.
cobra	The backend requires a stable Command Line Interface (CLI).	Viper[27] is a well-known configuration solution. It has wide support and was recommended. CLI
viper	Building and parsing application configuration.	Same reasons apply as with cobra[28].

Table 6.2: Libraries & Frameworks

Chapter 7

Architecture

This chapter presents the system architecture, summarizing the System Context Diagram (C1)[24]. To present a clear overview of the general flow only C1 was choosen. It provides a clearer picture of how a mission control operator is expected to interact with the bachelor's thesis solution. The second part discusses design decisions regarding the architecture. It again provides only an overview and a general direction of this bachelor's thesis' solution. This way, the initial design decisions won't be constraints during the development phase, as it allows for high flexibility.

7.1 System Context diagram

The System Context Diagram outlines the system's interactions with external entities, providing a high-level view of its scope and relationships.

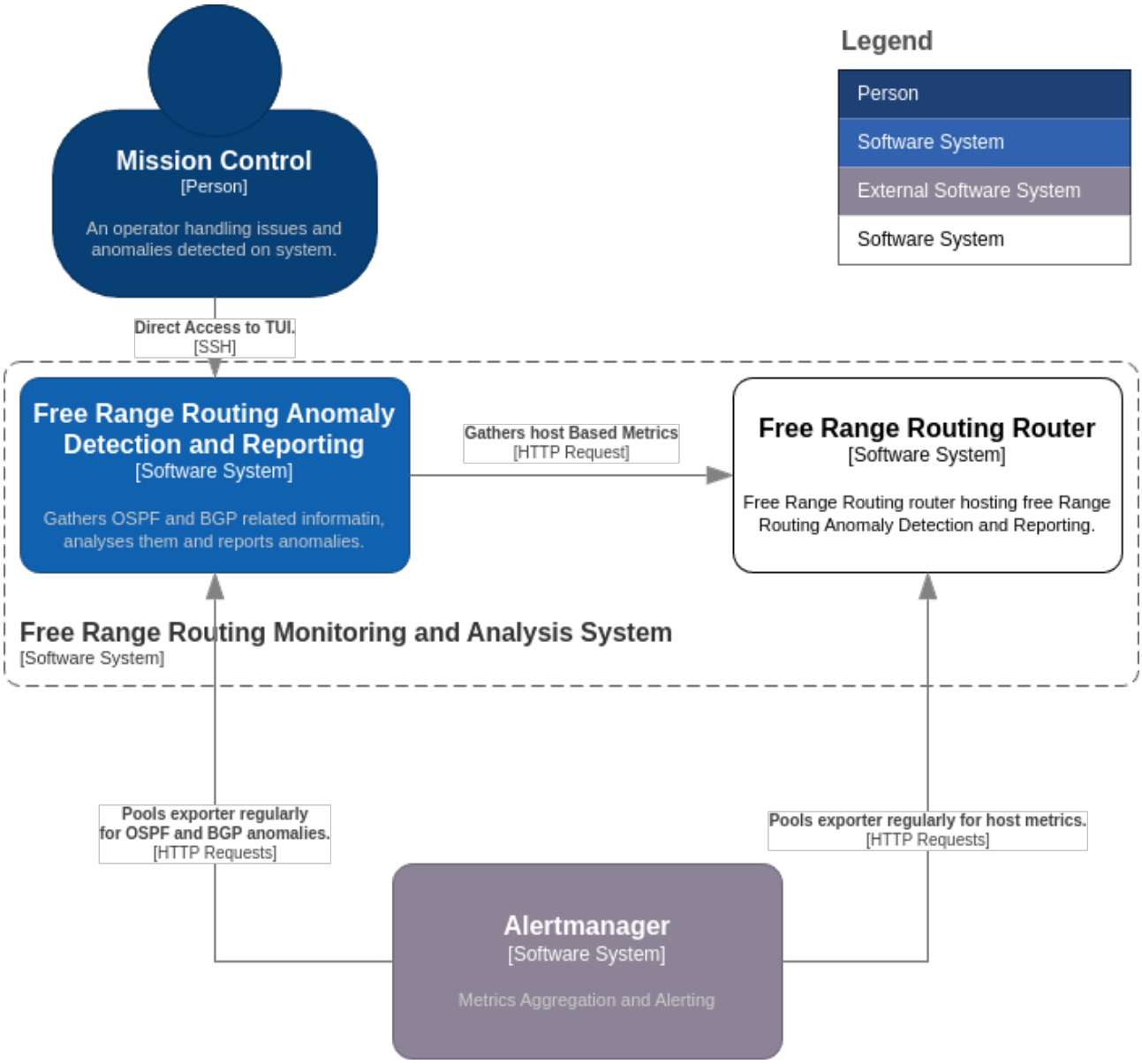


Figure 7.1: C1 System Context

7.2 Strategic Design

This section defines the problem domain using a ubiquitous language that all parties can easily understand. A solid design provides flexibility, domain prioritization and clear communications with all involved stakeholders.
Context Mapping describes the relationships of the different bounded contexts. It consists of the core domain, described as Free Range Routing Anomaly Detection and Monitoring. The composite bounded context is decomposed into two more granular bounded contexts[20].

Bounded Context	Type	Description
Free Range Routing Anomaly Monitoring	System	Allows the user to view issues related to OSPF and BGP anomalies. The user can interact with the interface to get further information or even solve anomalies.
Free Range Routing Anomaly Detection	System	Working in the backend as the core component of the system. Receives runtime configuration from the aggregation system and static configuration from the host system, recognizing and reporting anomalies to the monitoring system.
Free Range Routing Runtime Configuration Aggregation	Feature	A small aggregator gathering Free Range Routing related runtime configurations such as received routes from neighbors and reports them to the Anomaly Detection component.
Free Range Routing Anomaly Exporter	Feature	A Feature responsible for exporting detected Routing Anomalies.

Table 7.1: Bounded Context: Free Range Routing Anomaly Detection and Monitoring

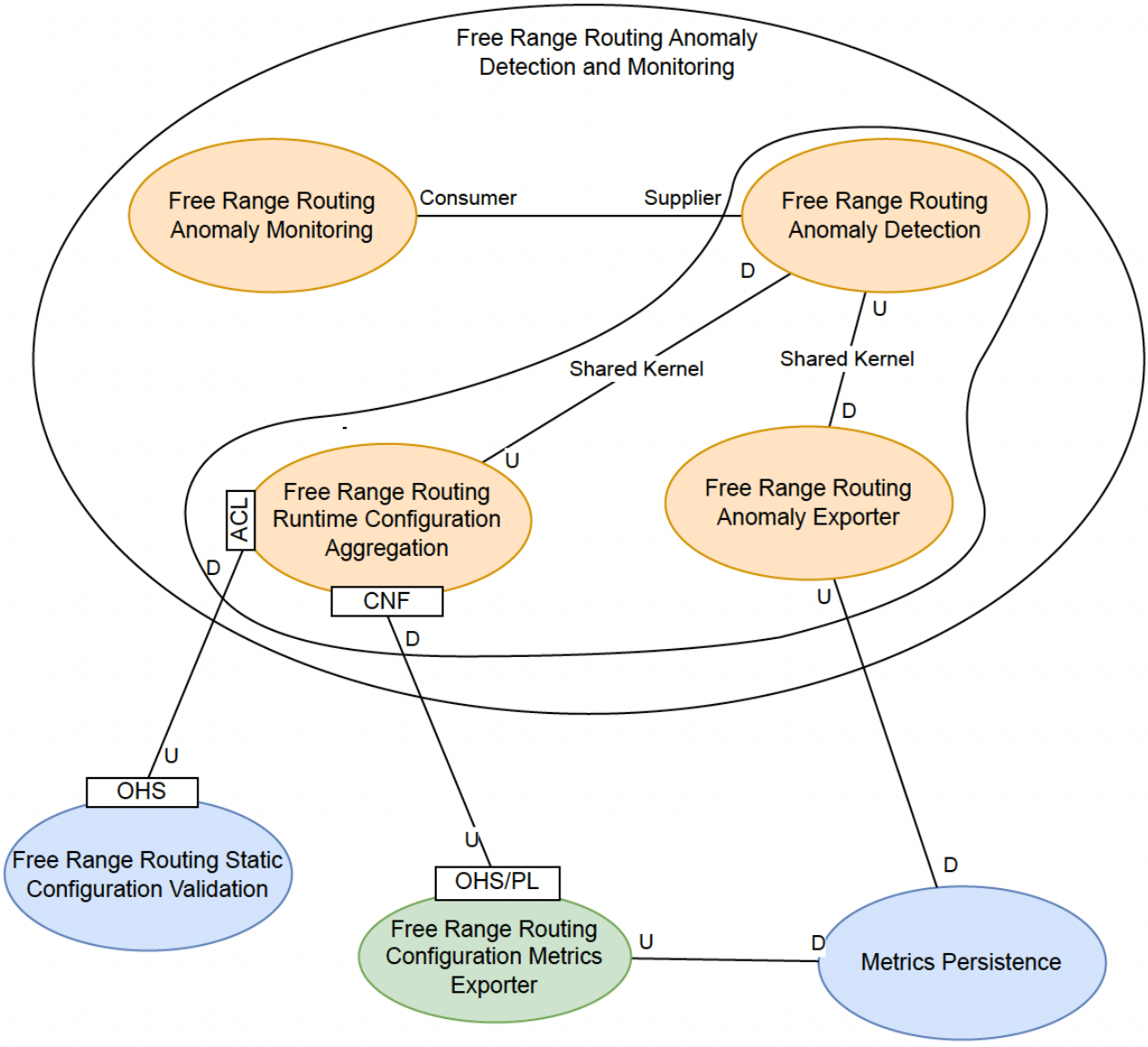
Bounded Context	Type	Description
Free Range Routing Configuration Metrics Exporter	System	A system that exports Free Range Routing specific host information.
Metrics Persistence	System	A system that aggregates information from metric exporters.

Table 7.2: Supporting Bounded Context

7.2.1 Adjustment Considerations

During the development process the accumulated knowledge lead to adjustments to the context map. Initially the core system required the two supporting context

- *Free Range Routing Static Configuration Validation* and

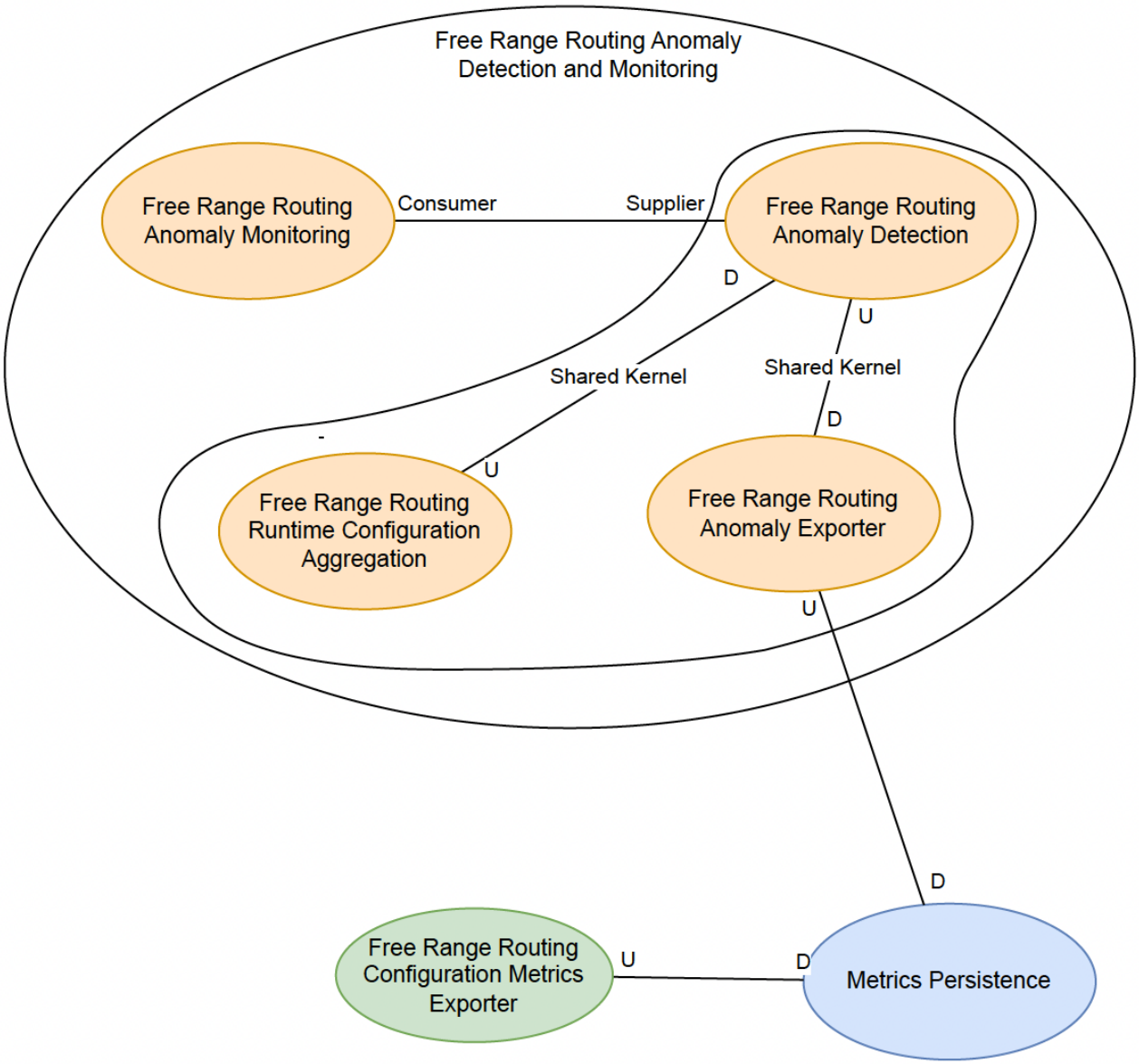


Legend: ACL - Anti-Corruption Layer, CNF - Conformist, OHS - Open Host Server, PL - Published Language, U - Upstream, D - Downstream, Orange - New Feature Addition, Blue - Existing Systems (Unchanged), Green - Existing Systems (To Be Modified)

Figure 7.2: Context Mapping Free Range Routing Anomaly Detection and Monitoring v1.0

- *Free Range Routing Configuration Metrics Exporter*.

The former proved impractical to implement, while the latter offered few benefits but imposed many restrictions on the flexibility of this application. Thus it has been decided to remove both these contexts.



Legend: ACL - Anti-Corruption Layer, CNF - Conformist, OHS - Open Host Server, PL - Published Language, U - Upstream, D - Downstream, Orange - New Feature Addition, Blue - Existing Systems (Unchanged), Green - Existing Systems (To Be Modified)

Figure 7.3: Context Mapping Free Range Routing Anomaly Detection and Monitoring v1.1

The decision to modify the context map at an advanced stage of the project was made following a comprehensive evaluation. Subsequent documentation delineates the benefits and drawbacks, concluding that this architectural refinement directly supports the intended functionality of the final solution.

Initially, the Validation system was intended as a supporting context. But the implementation of said system proved to be impractical. It requires a seperate server to run and adds complexity to an

already complicated system. Thus, it has been decided to remove this context and integrate it directly into the core system.

Free Range Routing Static Configuration Validation

As previously described, the complexity of this system proved inadequate, thus it has been removed. The subsequent list provides considerations regarding this removal.

Advantages

- Less complexity as the solution runs standalone.
- Less complexity as the solution doesn't require an additional data access layer between the core system and the Validation system.

Disadvantages

- Implementation of a parsing system is required.
- Reduced functionality as the previous system had a wide range of capabilities.

Free Range Routing Configuration Metrics Exporter

The Exporter still has its use, which is why the context has not been removed. However, the connection to the core system has been removed. This allows for independent development of said system and improvement in its functionality.

Advantages

- No reliance on the exporter for information gathering.
- Focused development of the core system.
- Respecting a stakeholder concern to reduce label cardinality concerning the Exporter system.

Disadvantages

- There are no notable disadvantages in this context. The solution already encompassed an extension to the Exporter system. This decision only shifted the implementation from the Exporter system to the core system.

Chapter 8

Quality Measures

8.1 Documentation

This section outlines our documentation standards and procedures to ensure consistency and quality across all project documentation artifacts.

8.1.1 LLM Usage

Throughout the documentation process, we make moderate use of AI language models such as ChatGPT and Claude. These tools primarily assist in improving spelling and grammar. This approach enhances the overall text quality by improving coherence and cohesion while maintaining the technical accuracy of the content[21],[7].

8.1.2 Documentation Principles

- All documentation is written in English.
- Abbreviations are permitted where contextually appropriate; otherwise, terms must be written in full.
- Redundancy is minimized and only allowed for demonstration purposes. Example: Describing a piece of code in two different chapters for comparative analysis.
- The main branch serves as the working branch, and all changes undergo compilation before being pushed to the repository.

8.1.3 Member Participation

- Each project member is required to actively contribute to documentation.
- All members must adhere to the established technology stack and documentation rules.

8.1.4 Documentation Context

- The documentation comprises two distinct parts:
  - **Technical Report**: Describes implemented features and development processes, including:
    * General issue description
    * Documentation breakdown
    * Tools and terminology
    * Present issues and solutions
    * Results and outcomes
    * Achieved requirements
    * Implementation details
    * Conclusion and future work
  - **Project Documentation**: Outlines guidelines and project journey, covering:
    * Requirements specification
    * Domain analysis
    * System architecture
    * Quality measures
    * All other miscellaneous elements
- All objects must be properly labeled, including:
  - Titles of all kinds
  - Images and figures
  - Tables and data representations
  - Code listings and references

8.1.5 Documentation Guidelines

- We utilize GitLab for source control management, shared documentation, and automated documentation building.
- Documentation is written in LaTeX, maintaining plain-text source files that compile to PDF using a LaTeX interpreter.
- For spellchecking, we implement Aspell for basic verification, integrated into our GitLab pipeline:
  - Mac: Aspell installation via Homebrew
  - Windows: Binary download from official sources
- A Makefile ensures consistent building processes across all team members' environments.

8.1.6 Document Guidelines

Having document guidelines helps us maintain document integrity when multiple parties work simultaneously. We push all updates directly to the main branch to ensure continuous visibility of modifications rather than confining changes to separate branches that only become visible near completion.

We implement basic commit rules for documentation, as overly complex guidelines could discourage frequent commits. These rules include a basic commit message structured by chapter name followed by a description of the change.

To enhance consistency, titles will be capitalized, and quick Aspell spell checks will be conducted when adding content. These practices improve the editing process and minimize verification and correction time.

As we use many acronyms and technical terms, we rely heavily on a glossary using the glossaries package, which includes TOC and acronym functionality. Glossary entries follow a specified template, and new entries must strictly adhere to the defined structure.

In summary, adhere to these guidelines:

- Titles must be capitalized.
- Use Aspell for manual error correction → `make interactive-spellcheck`.
- Our documentation uses no git branches to reduce project complexity.
- For in-document and out-document references, use \hyperref and \href.
- Simple commit rule: <chapter name>: <description of change made>.
- Use \gls{} or \glspl{} for glossary references.
  - \gls{} Print the term in lowercase.
  - \Gls{} Print the term in uppercase.
  - \glspl{} The plural form of the previous commands.
  - \Glspl{} The plural form of the previous commands.
- \acr
  - Use \acrfull{} the first time in a new chapter
  - Use \acrshort{} the first time in a new section
- For code snippets, use the package lstlisting, see example
- For tables of all kinds, use xltabular, which provides good support for longer tables, captions, and labeling without nesting.

- Use complete sentences in all itemizations and enumerations whenever possible. Apply standard punctuation rules.
  - When an itemization serves as a straightforward list, the final entry must conclude with a period (Example).

LaTeX Documentation Examples

```
% Note the distinction between Technical and General
\newglossary[glg]{general}{gls}{glo}{General Terms} % General terms glossary
\newglossary[alg]{technical}{als}{alo}{Technical Concepts} % Technical terms
    glossary

\newglossaryentry{openShortestPathFirst}{
    type=general,
    name={Open Shortest Path First (OSPF)},
    description={A dynamic routing protocol based on link-state technology,
        utilizing Dijkstra's algorithm to determine the shortest path. OSPF
        is widely used within autonomous systems for efficient IP routing.}
}

\newglossaryentry{ospfNeighbors} % Entry for technical glossary
{
    type=technical,
    name={OSPF Neighbors},
    description={OSPF routers form neighbor relationships with adjacent
        routers on the same network segment.
    Neighbors progress through several states (Down, Init, 2-Way, Full)
        before achieving full adjacency}
}

% Acronym for NFR
\newacronym{NFR}{NFR}{Non-Functional Requirement}
```

Listing 8.1: Example of new glossary entry

```
\href{https://foobar.com/}{\underline{Name of Reference}}

\hyperref[in-document-label-reference]{Name of in-document Reference}
```

Listing 8.2: Example of hyperref and href

```
% Example 1: Table with header
\begin{xltabular}{\textwidth}{|P{2cm}|R|}
  \caption{Example xltabular table} \label{tab:example-table} \\
  \endlastfoot
  \hline
```

```
  \rowcolor{headercolor}
  \myheadercell{Header 1} & \myheadercell{Header 2} \\
  \hline
  \endhead
  foo & bar \\ \hline
  foo & bar \\ \hline
\end{xltabular}

% Example 2: Table without header and background colored rows
\rowcolors{0}{rowcolor}{white}
\begin{longtable}{p{3cm} p{\dimexpr\linewidth-3cm-2\tabcolsep\relax}}
    foo & bar \\
\end{longtable}
```

Listing 8.3: Example of xltabular

```
% Example 1: Itemization with complete sentences
\begin{itemize}
    \item The documentation requires thorough spell correction.
    \item Documentation guidelines have to be thorough, to uphold consistency.
\end{itemize}

% Example 2: Itemization as a simple listing with final period
The features needs to support
\begin{itemize}
        \item e-mail verification
        \item password and,
        \item multi factor authentication.
\end{itemize}
```

Listing 8.4: Example of itemization

```
\lstinputlisting[
    language=go,
    caption={Example of itemization},
    label={lst:example-itemization}
]{03_project-documentation/resources/05_quality-measures/code-snippet-example}
```

Listing 8.5: Code Snippet Example

8.2 Development

This section outlines the tools and techniques we use to develop, test, and maintain our code. The TUI development represents a fresh start. To ensure future developers have an easy onboarding experience, we will deploy common and up-to-date tools and development techniques.

8.2.1 Code Guidelines

For new code development, we implement clean code guidelines as defined in Robert C. Martin's book "Clean Code"[16].

The structure of new files follows basic rules for vertical and horizontal formatting. To maintain readability, we limit file length to approximately 200 lines. Similarly, to prevent horizontal scrolling and improve code clarity, we target a maximum line length of 120 characters, with 80 characters being the preferred length.

With existing implementations, we adhere to clean code principles where practical, with the option of major rewrites. For new feature additions and modifications to existing code, we gradually align with these standards while respecting the current architecture.

Beyond these general guidelines, we have specific standards for our Go-based project:

- **Use Go modules** for dependency management to ensure reproducible builds.
- **Follow standard project layout** with /cmd, /pkg, and /internal directories to organize code.
- **Write thorough tests** with the standard testing package, aiming for good coverage.
- **Use meaningful error handling** rather than ignoring errors or panicking.
- **Implement consistent naming conventions** camelCase for variables, PascalCase for exported items.
- **Keep functions small and focused** on a single responsibility.
- **Use linters** like golangci-lint to catch common issues automatically.
- **Apply consistent formatting** with gofmt or go fmt.
- **Avoid global variables** as they make testing difficult and create hidden dependencies.
- **Use interfaces appropriately** to enable mocking for tests and increase flexibility.
- **Implement proper context handling** for cancelable operations.
- **Use defer for cleanup operations** like closing files and database connections.

8.2.2 Code Tools

- **GoLand (IDE)**: A specialized IDE for Go development that integrates built-in debugging, code refactoring, import completion, linting, and numerous other productivity features. As an IDE specifically designed for developing high-quality Go code, it delivers an optimal development experience.
- **VSCode/Codium**: Similar to GoLand, these are comprehensive IDEs that offer extensive tools to support efficient coding workflows and development cycles.

8.2.3 Code Review Guidelines

Once features are implemented, they need to be reviewed before merging:

- Feature implementations must be small, allowing for thorough code review and adherence to coding guidelines.
- Features must pass the testing and linting tools mentioned in the frr-tui development branch.
- Code test coverage must meet the requirements.
- A manual review process and subsequent approval by a second developer is required.

Feedback will be communicated to the developer. Once all necessary changes have been applied and the merge requirements are met, the feature is ready to be merged into the trunk.
Once merged, stakeholders can oversee progress and test the new features.

8.2.4 Environment

For faster and more inclusive development hot module reloading is implemented. This feature relies on a containerlab environment that is included in the FRR-MAD development repository.

To assure compatibility with the containerlab environment, the Proxmox Virtual Environment (PVE) is used as a remote development host. With modern IDEs like VSCode or Goland every developer is able to remote develop on a development virtual machine.

8.3 Testing

This section presents our testing strategy to ensure the quality and performance of all implemented FRs and NFRs .

8.3.1 Scope of Testing

Test Types

- **Automated Regression Tests:** Ensures that the application continues to perform as expected after changes or enhancements.
- **Cross-terminal and Cross-platform Tests:** Ensures consistent operation across different terminal emulators and operating systems.

Test Levels

We implement the testing levels presented in various software engineering courses:

- **Unit Testing:** Initial testing of individual code units in isolation.

- **Integration Testing:** Tests combinations of units for functionality, performance, and compatibility.
- **Acceptance Testing:** Validates complete application scenarios from an end-user perspective.

Test Environments

- **Development Environment:** Remote dev host environment utilized by developers during the implementation process.
- **GitHub Pipeline:** Configured for running automated tests after each commit.
- **Go Test Suite:** Employed for sub-project wide automated and integrated testing.

Tools and Technologies

- **Unit Testing:** Utilizing Go's built-in testing framework.
- **Automated Testing:** Integration with GitLab CI/CD pipeline.

Roles and Responsibilities

Key roles defined:

- **Developers:** Responsible for implementing unit and integration tests.
- **Tester:** Manages test environments and executes test plans.
- **Architect Lead:** Oversees the overall architecture of the testing environment, not necessarily the implementation details.

Testing Schedule

We follow an agile development methodology with no fixed testing schedule. Feature implementations are structured as user stories within two-week Scrum sprints. No merges are permitted into the main feature branch until the complete test suite executes successfully. Testing occurs continuously with every push to the repository via GitHub Actions.
Many of the details in this subsection are based on the technical report [17].

8.4 Quality Assurance

This section outlines our quality assurance procedures to ensure consistent delivery of high-quality software.

8.4.1 Definition of Done

The Monitoring and Anomaly Detection Tool is considered done when the following criteria are met:

1. **Functional Requirements Completed:**
   - The implementation of the MVP product has to be realized.
2. **Non-Functional Requirements:**
   - Implementation of all feasible non-functional requirements (NFRs).
   - NFRs are considered infeasible for implementation when their corresponding functional requirements are not implemented.
3. **Code Quality and Documentation:**
   - Code has been peer-reviewed, with no critical bugs or security vulnerabilities.
   - Unit and integration tests cover all relevant aspects of the backend.
   - Documentation is complete, including:
     - Technical documentation for developers.
     - User guide explaining how to install and use FRR-MAD.
4. **Testing and Verification:**
   - All tests (unit, integration, and acceptance) have passed in both development and production environments.
   - Codecov is utilized to track test code coverage.
5. **Stakeholder Sign-Off:**
   - Stakeholder has reviewed and approved the functionality of FRR-MAD.
   - Final sign-off has been obtained from both the development team and project manager.

8.4.2 Version Control Guidelines

In our project, we use the Trunk-Based Development approach to version control management. This strategy involves merging small, frequent updates directly into our main branch - the trunk. In our case, the trunk is named staging, which is kept up to date and always in a working state. Features will be introduced to the trunk once working. After intensive testing and a review process, the trunk will be merged into main.

Branches should be created as follows: <branch prefix>/<jira ticket number>-<concise branch name>
The possible prefixes are:

- feature: used for developing new features
- fix: used to fix bugs in the code

- experimental: used for new ideas or prototypes that should not be part of a release

For example:

- feature/FRR-1-Example-Task

Finally, we use semantic commit messages to add coherence when pushing changes. The format looks like this: <type of change>: <description of change>

The possible types of change are:

- feat: a feature was added to the code or is in development
- fix: something in the code was fixed
- style: formatting, missing semicolons, etc. - no production code change
- refactor: refactoring production code, for example, renaming a variable
- test: adding missing tests, refactoring tests; no production code change

With commit hooks this is automatically enforced. Every user will execute this script to enable this commit hook.

```
#!/bin/sh

mkdir -p .git/hooks

cat > .git/hooks/commit-msg << 'EOF'
#!/bin/sh

# Get the commit message from the first argument
commit_msg_file=$1
commit_msg=$(cat "$commit_msg_file")

# Define the allowed prefixes
valid_prefixes="feat:|fix:|style:|refactor:|test:"

# Check if the commit message starts with one of the allowed prefixes
if ! echo "$commit_msg" | grep -E "^($valid_prefixes)" > /dev/null
then
  echo "Error: Commit message must start with one of the following prefixes:"
  echo "Merge "
  echo "feat: "
  echo "fix: "
  echo "style: "
  echo "refactor: "
  echo "test: "
  exit 1
fi
```

```
27 EOF
28
29 chmod +x .git/hooks/commit-msg
```

Listing 8.6: Commit hooks setup

Chapter 9

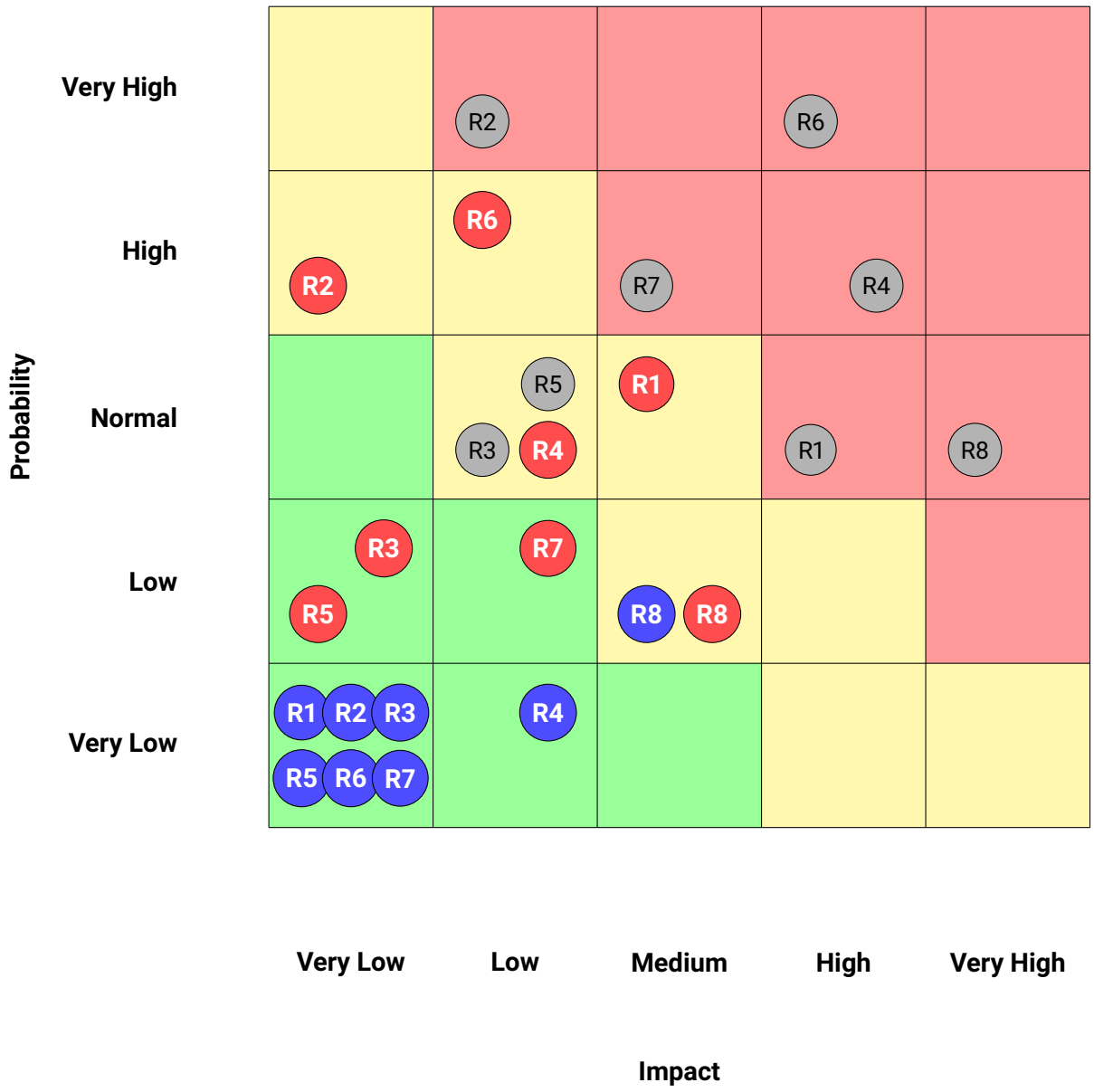
Risk Assessment and Mitigation

This section outlines the potential risks associated with the implementation of the monitoring tool for Free Range Routing within a Proxmox-based virtualized environment. The risks are categorized based on likelihood and impact, followed by corresponding mitigation strategies.

9.1 Version History

- **v1.0** - Initial risk assessment
- **v2.0** - Updated risk status based on operational experience

9.2 Risk Matrix



Explanation:

- **High Impact** - Directly affects core functionality or operational viability.
- **Moderate Impact** - Reduces efficiency, requires rework, but core functions remain operational.
- **Low Impact** - Minor inconvenience, cosmetic, or low-sensitivity issue.

Mapping to matrix:

- **Rare** - Highly unlikely to happen in the project's context.
- **Unlikely** - Can occur but only under specific corner cases.
- **Possible** - Plausible during normal development/operations.
- **Likely** - Expected during the project lifecycle.

Risk visualization:

- **Gray circles** - Pre-mitigation risk assessment showing initial risk positions (v1.0)
- **Red circles** - Post-mitigation risk assessment showing how risks have been reduced through appropriate control measures (v1.1)
- **Blue circles** - Current operational status (v2.0)

9.3 Risk Identification

Risk ID	Risk Description	Likelihood	Impact	Conditions
R1	Non-reproducible bugs due to inconsistencies in virtualized network behavior	Possible	High	• Virtual NIC driver variability • Proxmox/KVM virtualization inconsistencies • Packet processing differences
R2	Performance overhead from virtualized network stack causing inaccurate monitoring data	Unlikely	High	• High CPU/Memory load • Packet delays in virtual switches • Misconfigured resource allocation
R3	Data corruption or loss in virtual machine snapshots affecting monitoring results	Unlikely	Moderate	• Snapshot issues • Unstable Proxmox storage • Backup/restore conflicts
R4	Misconfigured virtual network topology leading to inaccurate routing data	Possible	High	• FRR config errors • VLAN misconfiguration • Bridge errors
R5	Incompatibility between monitoring tools (Prometheus, Grafana) and Proxmox environment	Unlikely	Moderate	• Kernel restrictions • Proxmox API limitations • Insufficient logging granularity

Risk ID	Risk Description	Likelihood	Impact	Conditions
R6	Docker container instability for FRR versions (e.g., FRR 8.5.4) leading to unpredictable behavior	Possible	High	• Version-specific bugs • Networking stack differences in Docker • Missing capabilities in container
R7	Deployment failure in Containerlab due to incorrect topology definitions	Possible	Moderate	• Wrong container mapping • Invalid connection definitions • Missing Docker images
R8	Human error during configuration and usage	Likely	High	• Manual misconfiguration • Insufficient training • Wrong network parameters

Table 9.1: Risk Identification Table

9.4 Risk Mitigation

Risk ID	Mitigation Strategy	New Risk Assessment
R1	Replace VM-based deployment with Containerlab and Ansible, reducing dependency on snapshots entirely.	Reduced likelihood to **Unlikely**
R2	Allocate dedicated CPU/Memory resources for Containerlab and apply Docker resource limits per container.	Reduced impact to **Rare**
R3	Configuration fully automated using Ansible and Containerlab, avoiding manual snapshot reliance.	Reduced likelihood to **Rare**
R4	Use pre-defined Containerlab topology files and version-controlled configurations, avoiding manual editing.	Reduced likelihood to **Unlikely**
R5	Perform compatibility testing for monitoring stack (Prometheus, Grafana) within Containerlab before deployment.	Reduced likelihood to **Rare**

Risk ID	Mitigation Strategy	New Risk Assessment
R6	Pin Docker image versions to tested stable versions (e.g., FRR 8.5.4) and add custom health checks to detect container misbehavior.	Reduced likelihood to **Unlikely**
R7	Validate topology files with CI pipeline before deployment to Containerlab. Add schema validation tools to catch errors.	Reduced likelihood to **Rare**
R8	Implement "least privilege" access model and automate configuration using Ansible, reducing manual steps.	Reduced likelihood to **Unlikely**

Table 9.2: Risk Mitigation Strategies

9.5 Risk Status Update (v2.0)

Table 9.3: Risk Status Update

Risk ID	Previous Status	Current Status	Change Reason
R1	Possible/High	Rare/Medium	No occurrences observed in production VMs
R2	Unlikely/High	Mitigated	Sufficient resources confirmed in all environments
R3	Unlikely/Moderate	Mitigated	No cases of data corruption detected
R4	Possible/High	Rare/Medium	Stable topology confirmed through testing
R5	Unlikely/Moderate	Mitigated	Full compatibility verified in testing
R6	Possible/High	Mitigated	No stability issues found in production
R7	Possible/Moderate	Rare/Low	Test coverage confirms mitigation not required
R8	Likely/High	Unlikely/Medium	No change - monitoring remains in place

Chapter 10

Testing

10.1 Unit Tests

Developers are expected to write unit tests for their code, when appropriate and meaningful. The developers are encouraged to use the principles of TDD, meaning to write the Unit Tests in advance to ensure a form of "safety net" and immediate feedback whether the code's behavior is unexpected. This approach also ensures there is no regression, where new features break previously existing functionality.
The use of Unit Tests will mainly occur in the backend side, since Unit Testing the frontend would not make sense in this particular case. The frontend will be tested using Acceptance Tests.
Side note: Every developer is informed that using TDD isn't a mandatory approach, but more of a suggestion. If by chance the developer decides that writing Unit Tests in advance for a specific feature would be disadvantageous, he is free to approach the development as he pleases.

10.2 Acceptance Testing

Acceptance testing is a type of software testing conducted to determine whether a system meets the functional requirements. It serves as a final verification phase to ensure that the developed features behave as expected from the user's perspective. In our project, acceptance tests were performed by our development team, the advisor and the stakeholder. This process helped confirm that the implemented functionalities aligned with the agreed-upon requirements and that the system fulfilled its intended purpose.

10.2.1 Testing Plan

Who	• QA-Team: The QA-Team consist of all three students of this thesis. • Advisor: • Project Partner:
When	The Acceptance test will be execute two times: • After the MVP was developed (exclusively by the development team). • Two weeks before the final submission. This provides an opportunity to identify any misunderstandings or errors in the MVP early on, allowing ample time for corrections. By conducting the second acceptance test shortly before submission, we can still implement fixes for any optional features.
What	During the first acceptance test, we test the Functional Requirements (FRs) included in the MVP The Second run will cover all FRs we accomplished

Table 10.1: Acceptance Testing Plan

The detailed testing protocol is documented in the Appendix, which is crucial to ensure a standardized, consistent testing process and the reproducibility of results.

Chapter 11

Project Plan

11.1 Resources

This section evaluates the available resources required for implementing the moniotring tool.

11.1.1 People

Name	Skills
Mino Petrizzo	Experienced in Web Developmen going in direction Cyber Security Good Knowledge in PHP, Python, Golang
Roman Cvijanovic	Strong background in system engineering with experience from the ISP sector Skilled in containerization and infrastructure automation using Proxmox, Docker, and Ansible
Yannick Staedeli	Experienced in/with Network and Cloud Engineering / Telecommunication Systems Fundamental Knowledge of JavaScript, HTML, CSS, Java, SQL, Python, Golang

Table 11.1: SCRUM Role distribution

11.1.2 Time

The Bachelor's Thesis officially started on **February 17, 2025**. Within the first week, it is necessary to hold a kickoff meeting with the advisor and eventually sign the official assignment. Although a Normal Semester has 14 weeks, our spring term has 15 weeks plus 1 week spring break plus 1 week extra time for the bachelor's thesis. Therefore, the final submission deadline is set for **June 13, 2025**.

To estimate the time, we can invest into this project, we take the official time that is recommended for 12 ECTS Credits. That is 360 hours per person. This gives us a total of **1080** hours to meet all of our deadlines, with 40 hours per person allocated for the extra week. If we distribute the remaining time frame of 960 hours evenly across the 15 school weeks dedicated to the project, we will have 64 work hours per week as a team. This is about 21.3 hours per person per week. Over the course of a week, we will spend an average of 2.25 hours on fixed project-related meetings and 1.5 hour conducting and preparing for the meeting with our advisor or other stakeholders. This leaves 17.55 hours for each of us to work alone or in pairs on specific project-related tasks.

11.2 Roles

Accordingly, we have defined the following roles and divided them among ourselves. These roles are the basic project management roles (SCRUM) and the typical software development roles necessary for good, qualitative products.

11.2.1 SCRUM Role Distribution

Role	Members	Explanation
Product Owner	Mino Petrizzo	As the product owner, Mino Petrizzo will prioritize the direction of the development of the product..
Scrum Master	Roman Cvijanovic	As the Scrum Master, Roman Cvijanovic will lead the Sprint Meetings, keep an overview of feature progress and keep the team informed.
Developer	Yannick Staedeli Mino Petrizzo Roman Cvijanovic	As developers, everyone is invested to develop a product that satisfies current standards and the Stakeholder's requirements.

Table 11.2: SCRUM Role distribution

11.2.2 General Role Distribution

Role	Members	Explanation
Architect/System Design Lead	Roman Cvijanovic	A very important decision, as this pertains the backend and frontend.
Tester	Mino Petrizzo	The tester is a clear-cut role. Their task is to regularly to through our test scenarios of check them as fulfilled.
Customer	OpenSystems	The customer is the recipient of our solution. Their opinion has the highest sway over our decisions.
Advisor	Severin Dellsperger	The advisor is our first contact and bridge to the customer stakeholder. Additionally they will, as their role implies, advise us in case of sub-optimal decisions.
Second Reader	Olaf Zimmermann	The second reader will hold no active position during the project.
Project Leader	Severin Dellsperger	As the project Leader, Severin Dellsperger will be the tie breaker if the team finds no common ground.

Table 11.3: General Role distribution

11.2.3 Roles scope

It is important to keep track of the scope each role has to cover, so there will not be the problem of certain work being done double, and other work not being done at all. We have to clearly define, who takes care of what.

Product Owner

The scope of a Product Owner (PO) typically involves various responsibilities and activities throughout the product development life-cycle. Here are some key aspects of the scope of a Product Owner:

1. **Defining product vision:** The Product Owner is responsible for defining and communicating the overall vision for the product.
2. **Progress Checkups:** The product owner will check on our progress and ensure we are steering in the right direction.

Scrum Master

The scope of a Scrum Master involves various responsibilities and activities aimed at facilitating the Scrum framework's successful implementation and ensuring the team's effectiveness. Here are key aspects of the Scrum Master's scope:

1. **Facilitating Scrum Events:** The Scrum Master facilitates various Scrum events, including Sprint Planning, Daily Stand-ups, Sprint Reviews, and Sprint Retrospectives. They ensure that these events are conducted effectively, time-boxed, and focused on achieving their objectives.
2. **Removing Impediments:** The Scrum Master identifies and removes impediments that hinder the team's progress. This involves addressing issues such as organizational barriers, resource constraints, and conflicts within or outside the team.
3. **Coaching and Mentoring:** The Scrum Master coaches the Scrum Team and Product Owner on Scrum principles, practices, and values. They help individuals and teams understand and adopt Scrum roles, artifacts, and ceremonies.
4. **Journal:** The Scrum Master journals sprints and meetings in an open manner and shares everything transparently.

Developer

In this project, the Developer role encompasses multiple technical responsibilities. Instead of creating separate roles for closely related tasks, we've consolidated these responsibilities under the Developer role. This approach allows for flexible task distribution among team members throughout the project lifecycle.

1. **Code Development** Primary responsibility involves implementing features, writing maintainable code, creating comprehensive unit tests, and deploying completed functionality to the production environment.
2. **Testing** Due to our streamlined team structure, developers perform testing duties during Pull Request reviews at sprint boundaries. This includes integration testing, regression testing, and verifying acceptance criteria.
3. **Quality Assurance** Developers are responsible for code quality enforcement through peer reviews, static code analysis, and adherence to coding standards. This includes monitoring code coverage, identifying technical debt, and ensuring documentation completeness.

Architect/System Design Lead

The Architect Lead/System Design Lead has the important task of design the cohesion of the test system on which we will base our final product on. The complexity of our task requires the Architect Lead/System Design Lead to have profound expertise. The tasks pertain

1. **Test System Design:** Which should replicate a real world scenario based on FRR.
2. **Comprehensive System Architecture:** No missing components and all the requirements reflected in the architecture.
3. **Network Scheme:** Comprehensive network scheme based on real world scenarios to test our monitoring solution on.

Tester

The Tester holds an equally important job to the Architect Lead/Systemd Design Lead. At the end of the project our product should have a complete testing kit and still enable extensibility in this swift changing environment. The Tester's jobs include

1. **Test Environment:** Decide if a test environment fulfills the FRs and NFRs requirements.
2. **Test Reviews:** Be it either manual testing or automatic, all tests have to be tested, both bad routes and happy routes.
3. **Test Scope:** Together with the Architect Lead/System Design Lead the Tester will design the test scope, also based on FRs and NFRs.

Customer

The stakeholder is the main beneficiary of the solution of this project.

1. **Conveying wishes:** As the main beneficiary the stakeholder states the goals to be achieved mostly in an abstract manner.

Advisor

The Advisor is our main contact point for the customer. While we will stay in touch with the customer directly, our Advisor will be there to support us in case of any miscommunication happening.

Second Reader

The Second Reader's task consists of proof-reading our solution documentation.

Project Leader

The Project Leader is responsible for overseeing the overall direction of the project and ensuring that the team stays aligned with its goals. In cases of divergent opinions or conflicts during decision-making, the Project Leader holds the final say to resolve disputes and maintain progress.

11.3 Project Planning and Tracking

Managing task allocation and progress tracking in complex projects like our bachelor's thesis requires robust project management infrastructure. Our team has implemented JIRA as our primary project management tool. We selected JIRA based on its comprehensive feature set and the team's existing proficiency from prior development projects.
JIRA provides essential capabilities for our development workflow:

- Agile boards for sprint planning and execution

- Task tracking with customizable workflows
- Integration with version control systems
- Story point estimation and velocity tracking
- Automated reporting and metrics collection
- Configurable issue types and fields

The team's familiarity with JIRA's interface and functionality enables immediate productive use without additional training overhead. This standardization on JIRA streamlines our project management processes and facilitates effective sprint management.

11.4 Time Tracking

To provide an accurate time tracking, we use an additional app in Jira called "Timetracker". Every task for our project is documented within our issue management system Jira. This systematic approach logically extends to time tracking, where we record the duration spent on each issue. We must also account for the time spent in meetings, as it constitutes a significant portion of our project efforts. To facilitate this, we create an issues in the Backlog for meetings. Every participant is then required to log their time against this issue with a precise tag.

How to Tag the Time Logs

Ultimately, we aim to provide a precise overview of the hours invested in our project. This schema clearly illustrates how we tag our time logs. **Each time log must contain one tag from each color.** If we follow that principle, we can generate a chart based on a row of tags. Each row of tags in the scheme represents the total time spent on the project.

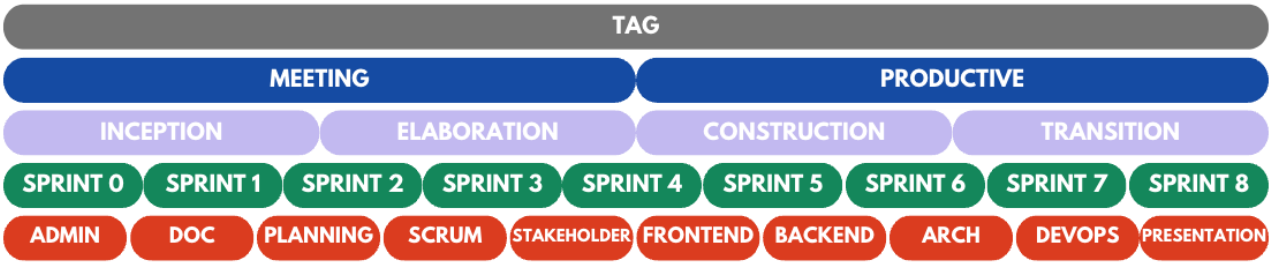


Figure 11.1: Time Tracking Tags

Tag Description

Admin: General activities related to our bachelor's thesis, including personal reports and coordination tasks not directly tied to the technical project work, but still within the academic scope.
documentation (doc): Writing technical documentation, usage guides, and final report sections related to the project.
Planning: Sprint planning sessions, long-term project planning, workload estimation, and other preparatory tasks.

SCRUM: Agile ceremonies including daily stand-up meetings, sprint retrospectives, and sprint reviews.
Stakeholder: Meetings and other forms of communication with our industry partner, Open Systems, including requirement gathering and feedback sessions.
Frontend: Implementation of the terminal-based user interface (TUI), focusing on usability features such as text highlighting and interactive command-line output.
Backend: Core logic and data handling functionalities of the FRR-MAD Tool, including OSPF data parsing, metrics processing, and integration with Prometheus.
architecture (arch): Tasks involving network lab setup, infrastructure design, architectural decision-making, and visual scheme creation.
DevOps: Activities related to CI/CD pipelines, automated testing, infrastructure-as-code, and deployment workflows for the project.
Presentation: Preparation of slides, speaker notes, and practice sessions for the mitderm and final project presentation.

Time Tracking Reports

The detailed time-tracking report is in the appendix, along with step-by-step instructions for generating your own reports in our time-tracking platform.

11.5 Project Schedule

To facilitate agile execution, the project is broken down into nine sprints, each focusing on incremental progress. Key milestones are strategically placed throughout the schedule to assess project status and ensure alignment with goals.

A visual timeline (see diagram) illustrates the relationship between phases, sprints, and milestone checkpoints, ensuring transparency in tracking development progress. Notably, milestone dates serve as key reassessment points, allowing for timeline adjustments if necessary.

Our plan is to continue development during spring break, which either reduces the weekly workload (20 hours per week instead of 21.3 over a 15-week schedule) or provides the opportunity to implement optional features. All additional calculations will be based on 20 hours per week per person, but the actual time dedicated to the project may vary.

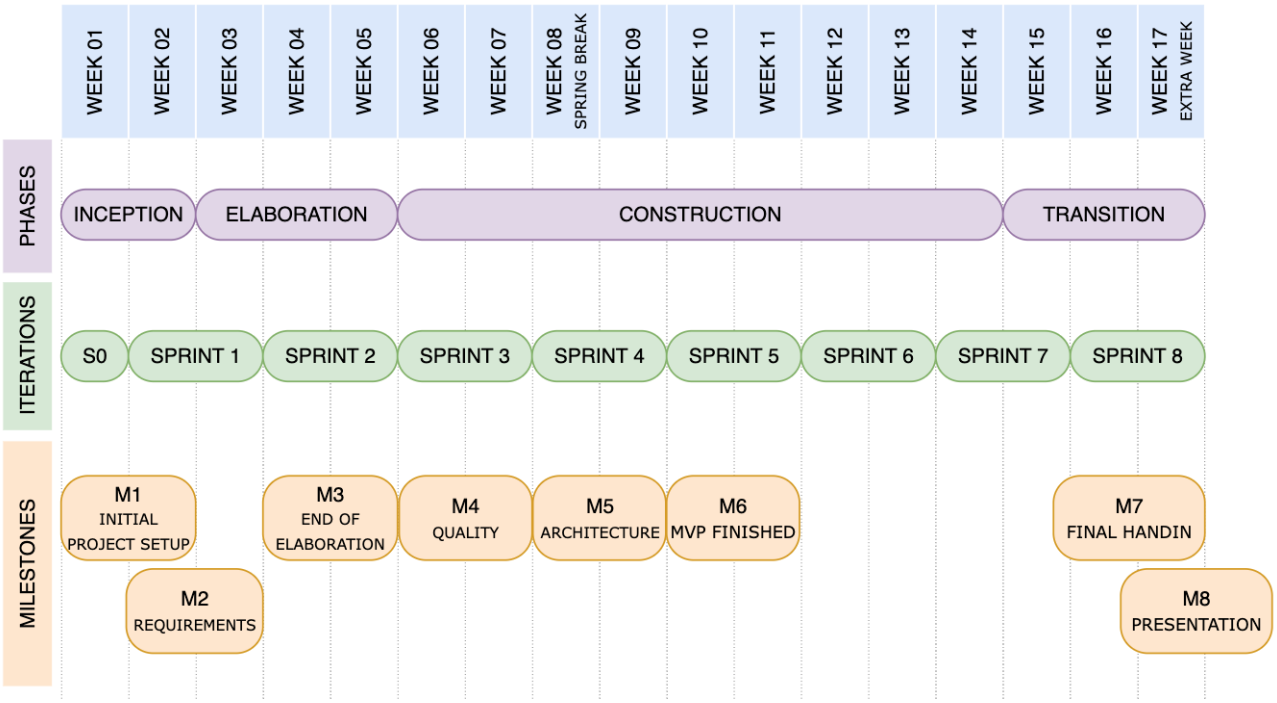


Figure 11.2: Schedule - Phases, Iterations and Milestones

11.5.1 Phases

The project can be divided into four distinct phases: inception, elaboration, construction, and transition, which align with the long-term planning principles of RUP.

Inception

In the inception phase, we shape an approximate vision for the project, focusing on the needs of our industry partner Open Systems. Therefore, a kickoff meeting with all stakeholders is necessary and will be held in week 01. During this stage, we define the scope by evaluating various features for the monitoring system based on the inputs from Julian Klaiber. This allows us to develop an initial understanding of the project's intricacies, accompanied by rough estimates for efforts to gauge the anticipated workload.

Elaboration

The Elaboration Phase focuses on refining the project scope, validating the system architecture, and assessing feasibility before full development begins. A sophisticated domain analysis is conducted to evaluate existing OSPF monitoring solutions and determine whether developing a new tool provides unique value. During this phase, we finalize functional and non-functional requirements, ensuring alignment with the industry partner's needs. The system architecture is defined, including data collection mechanisms and the TUI design, while key risks such as performance bottlenecks and hardware compatibility are assessed and mitigated through early testing. The development environment is set

up, including selecting programming languages, network simulation tools, and version control systems. A prototype is then developed to test core functionalities such as real-time OSPF state retrieval and visualization. Finally, the project plan and milestones are refined based on insights gained from initial testing, ensuring a solid foundation for the next development phase.

Construction

The Construction Phase is centered around the implementation of the FRR-MAD Tool based on the previously defined requirements. After finalizing the application architecture, the focus shifted to successfully completing the Minimal Viable Product (MVP). This includes implementing the core functional requirements such as exporting OSPF routing metrics to Prometheus (FR1-1), collecting real-time routing data from FRR (FR2), and enabling OSPF-specific anomaly detection within the terminal user interface (FR3 and FR4). Additionally, the MVP supports querying and displaying routing anomalies through the TUI (FR4), offering users a clear and interactive overview of network irregularities. Development proceeds in short iterations with continuous integration and regular validations against the defined requirements. Unit testing and manual inspections ensure functional correctness, while frequent reviews with our industry partner support alignment with practical use cases. By the end of this phase, the FRR-MAD Tool reaches a stable MVP state and additional optional features, ready for evaluation and deployment in the final project stages.

Transition

The Transition Phase marks the final stage of the project, focusing on preparing the FRR-MAD Tool for production use and deployment. During this phase, we collaborated closely with the stakeholder to conduct thorough testing, address any remaining issues, and ensure that all core functionalities met the defined expectations. Particular attention was given to identifying and resolving bugs, optimizing performance, and validating the system in a realistic operational environment.
In parallel, we implemented several optional features that were not part of the initial MVP but added practical value for end users. These enhancements were prioritized based on stakeholder feedback and technical feasibility. The deployment process was accompanied by final documentation, internal handover preparations, and knowledge transfer to ensure long-term maintainability of the system. Overall, the Transition Phase ensured a smooth and stable rollout of the tool, aligning technical outcomes with the project's strategic goals.

11.5.2 Iterations

The iteration focuses on the short-term planning of the project. For it, we will use the SCRUM methodology. Furthermore, the 2-week sprint cycle and regular meetings will expedite risk assessment, enabling prompt re-planning if necessary. Sprint Zero is deliberately limited to one week, as it primarily involves administrative tasks such as project setup and initial planning. This approach ensures that the foundational aspects of the project are established as quickly as possible, allowing us to begin development without delay.

11.5.3 Milestones

Finally, the milestones for our project will be defined based on high-level objectives, not specific epics. We try to follow this principle due to the high demand of agility in this project, and it enables reassessment and course correction. Given that these milestones have fixed dates, they are well-suited to serve as key points in our project timeline and will be utilized accordingly. Since Jira does not natively support milestones, we use the Releases feature to track them. This approach integrates well with our project tracking, as milestones are clearly represented in JIRA's timeline overview.

11.6 Processes and Meetings

This section aims to specify in more detail how we utilize the SCRUM methodology. Furthermore, it presents a plan outlining when each meeting will be held.

11.6.1 Processes

As we have already clarified the reason for using SCRUM+ in section Collaboration Framework, we now aim to define exactly how and what we will implement in our project. We want to use Scrum's flexible, iterative approach to organizing our tasks and prioritizing our work. We intend to apply the core principles of Scrum, including its defined roles and key artifacts such as the Product Backlog, Sprint Backlog, and Increment. In addition, we plan to incorporate selected Scrum events into our workflow.

Customizing to our own Needs

Based on our previous project experience, we recognize the value of daily and weekly Scrum meetings. While the standard practice is to hold a 15-minute daily meeting, we have adjusted this to three meetings per week to better fit our 20-hour workweek. This approach helps keep our team aligned and ensures steady progress. At the beginning of each sprint, we conduct Sprint Planning immediately after completing the Retrospective and Sprint Review of the previous sprint. By combining these meetings into a single session, we aim to make them more efficient, reduce overhead, and maximize our limited time.
For coordination, quick updates, and questions, we have created a Teams channel dedicated to developers and another channel that includes both developers and the advisor. Communication with stakeholders will take place through a dedicated Slack channel.

11.6.2 Meetings

We decided to create a plan for all of our meetings. The plan starts in Week 2 because we decided to shorten Sprint Zero (project setup) to one week.

Figure 11.3: Meetings Plan

Sprint Planning

Who: the entire team
When: at the start of each Sprint cycle
What: In the Sprint Planning we will define a sprint goal, select items from the product backlog, and assign story points to them. We will break up each one of these stories into smaller sub-tasks. This means that a separate Sprint Planning 2 meeting will not be held. This meeting will be prepared and led by the product owner.
How long: approximately 90 minutes

Project Elaboration

Who: the entire team
When: every Monday
What: In this meeting, all team members present their work from the past week. During this meeting, team members have the opportunity to provide more detailed explanations and share important information relevant to the project's progress and objectives.
How long: approximately 30 minutes

Daily SCRUM

Who: the entire team (if possible)
When: two times a week
What: In this meeting, we present small progress updates on our tasks, and it serves as an opportunity to ask for help if needed. This meeting does not require the participation of the entire team, and in some cases, up to three such meetings may be held in a week.
How long: approximately 15 minutes

Retrospective

Who: the entire team
When: at the end of a sprint
What: In this meeting, the team reflects on the previous sprint, discussing what went well, what could be improved, and any obstacles faced. The goal is to identify actionable improvements to enhance team performance in future sprints.
How long: approximately 15 minutes

Sprint Review

Who: all involved members (dev team, advisor, Open Systems representative)
When: after a completed Sprint
What: In this meeting, the team presents the completed work to stakeholders, demonstrating the implemented features and gathering feedback. It serves as an opportunity to assess progress, adjust priorities, and align the product with stakeholder expectations.
How long: approximately 20 minutes

Advisory Meeting

Who: the entire team and the advisor
When: every Thursday
What: In this meeting, the team presents and discusses the project's progress with the advisor. The goal is to receive feedback on the work completed and project planning to ensure alignment and continuous improvement.
How long: approximately 60 minutes

11.6.3 Estimated Time plan per Week

If we strictly follow our plan, this is how one person's work should be distributed over a week.

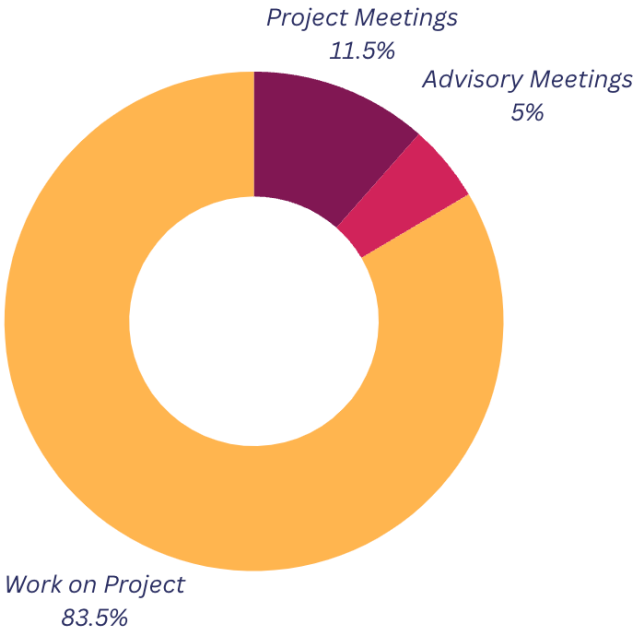


Figure 11.4: Work per Person per Week

11.7 Project Management and Development Workflow

11.7.1 Jira

Purpose: Jira is used for project planning, task management, and team collaboration.

Usage:

- Configured with custom flags to fit our project management needs.
- Used to create and track tasks for team members.
- Manages development timelines and prioritization of issues.

Bibliography

[1] Anarkiwi. google-quagga: Google fork of quagga routing software. https://github.com/anarkiwi/google-quagga, 2024. Last accessed: 2024-03-02.

[2] arc42 Team. arc42: Effective, lean and pragmatic architecture documentation and communication. https://arc42.org/documentation/. Template and methodology for software architecture documentation with 12 standardized sections; accessed June 12, 2025.

[3] BaiMeow. Networkmonitor. https://github.com/BaiMeow/NetworkMonitor, 2024. Last accessed: 2024-03-02.

[4] charmbracelet/bubbletea. Bubble tea: A powerful little tui framework. https://github.com/charmbracelet/bubbletea. Go framework for building terminal applications based on The Elm Architecture; accessed June 12, 2025.

[5] charmbracelet/lipgloss. Lip gloss: Style definitions for nice terminal layouts. https://github.com/charmbracelet/lipgloss. Go library for styling terminal applications with CSS-like syntax, tables, lists, and trees; accessed June 12, 2025.

[6] Cisco Systems. Open shortest path first (ospf) administrative distance. https://www.cisco.com/c/en/us/support/docs/ip/border-gateway-protocol-bgp/15986-admin-distance.html#toc-hId--1692530466. Explains AD values such as 110 for OSPF; updated Sept 27, 2024; accessed June 10, 2025.

[7] Claude. Claude: Conversational ai model and image generator. https://claude.ai/, 2024. Last accessed: 2024-12-19.

[8] Practical DevSecOps. Threat modeling vs risk assessment: Understanding the difference. https://www.practical-devsecops.com/threat-modeling-vs-risk-assessment/, 2024. Last accessed: 2024-11-18.

[9] Roman Dodin and Srlabs Team. Containerlab: Container-based networking lab. https://containerlab.dev/, 2024. Active open-source project (latest release: v0.68.0).

[10] Inc. Figma. Figma: Collaborative interface design tool. https://www.figma.com, 2024. Last accessed: 2024-11-25.

[11] Google Protocol Buffers Team. Protocol buffer basics: Go. https://protobuf.dev/getting-started/gotutorial/. Tutorial for using Protocol Buffers with Go, covering message definition, compilation, and API usage; accessed June 12, 2025.

[12] Craig Larman. Applying evolutionary requirements. https://www.craiglarman.com/wiki/downloads/applying_uml/larman-ch5-applying-evolutionary-requirements.pdf, 2002. Accessed: 2025-03-12.

[13] Craig Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004.

[14] JGraph Ltd and draw.io AG. diagrams.net (formerly draw.io): Online drawing tool. https://app.diagrams.net/, 2024. Last accessed: 2024-12-19.

[15] M. Mitchell, J. Dickinson, and G. Huston. Rfc 6996: Autonomous system (as) reservation for private use. https://datatracker.ietf.org/doc/rfc6996, 2013.

[16] Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Robert C. Martin Series. Prentice Hall, 2009.

[17] Livio Mauchle, Simon Hefti, Martyn Foreman, Yannick Städeli, and Luca Köppel. Swisscardgames: Final documentation. Technical report, School of Computer Science, OST Eastern Switzerland University of Applied Sciences, 2024.

[18] Nvidia Networking. Cumulus linux 4.1 - frrouting overview. https://docs.nvidia.com/networking-ethernet-software/cumulus-linux-41/Layer-3/FRRouting-Overview/, 2024. Last accessed: 2024-03-02.

[19] Juniper Networks. Juniper, configure ospf route control. https://www.juniper.net/documentation/us/en/software/junos/ospf/topics/topic-map/configuring-ospf-route-control.html. Last accessed: 2025-05-30.

[20] Olaf Zimmermann. Domain-driven design. Lecture: Application Architecture (AppArch), OST - Eastern Switzerland University of Applied Sciences. Course materials available at https://ozimmer.ch/; accessed June 12, 2025.

[21] OpenAI. Chatgpt: Conversational ai model and image generator. https://openai.com/chatgpt, 2024. Last accessed: 2024-12-19.

[22] OSRG. Gobgp: Open source bgp implementation. https://github.com/osrg/gobgp, 2024. Last accessed: 2024-03-02.

[23] Batfish Project. Batfish: Network configuration analysis tool. https://github.com/batfish/batfish, 2024. Last accessed: 2024-03-02.

[24] Simon Brown. The c4 model for visualising software architecture. https://c4model.com/. Hierarchical approach to software architecture diagramming with context, containers, components, and code levels; accessed June 12, 2025.

[25] SNAS. Openbmp: Bgp monitoring protocol collector. https://github.com/SNAS/openbmp, 2024. Last accessed: 2024-03-02.

[26] SoftwareTestingHelp. Top 15 network monitoring tools. https://www.softwaretestinghelp.com/network-monitoring-tools/, 2024. Last accessed: 2024-03-02.

[27] spf13/cobra. Cobra: A commander for modern go cli interactions. https://github.com/spf13/cobra. Go library for creating powerful modern CLI applications with subcommands, flags, and auto-completion; accessed June 12, 2025.

[28] spf13/viper. Viper: Go configuration with fangs. https://github.com/spf13/viper. Go configuration library supporting JSON, TOML, YAML, HCL, INI, envfile and Java properties formats; accessed June 12, 2025.

[29] Tynany. frr_exporter: Prometheus exporter for frrouting. https://github.com/tynany/frr_exporter, 2024. Last accessed: 2024-03-02.

[30] Vadims06. Ospf topology watcher. https://github.com/Vadims06/ospfwatcher, 2024. Last accessed: 2024-03-02.

List of Figures

List of Tables

List of Algorithms

Listings

Chapter A

Technical Resources

A.1 List of Tools and Resources

Field of Work	Tools
collaboration and project management	Teams, Slack, Outlook, Jira
coding	Visual Studio Code, VSCodium, JetBrains (GoLand, PyCharm, Writerside), ChatGPT, claudeAI, DeepSeek
data analysis and visualization	Diagrams (former draw.io), Canva
text creation, text optimization, spelling and grammar check,	Visual Studio Code, VSCodium, JetBrains (GoLand, PyCharm, Writerside), ChatGPT, claudeAI, DeepL, DeepSeek, latex, MacTex
DevOps	Ansible, GitLab, GitHub, Containerlab, Docker, Pipelines, Actions, Scripts, AIR - Hot Module Reloading (HMR)

Table A.1: Tools and Resources

Chapter B

Testing Reports

B.1 Acceptance Test Reports

B.1.1 QA Team

Table B.1: Acceptance Test Protocol - QA Team

Participant's Name	**Yannick Städeli, Mino Petrizzo, Roman Cvijanovic**
Date	May 09, 2025
Location	Remote
	Continued on next page

Table B.1 – continued from previous page

Participant's Name	**Yannick Städeli, Mino Petrizzo, Roman Cvijanovic**
Tasks	FR1-1: Export OSPF Routing Metrics → Participant Feedback: Metrics for installed and OSPF routes implemented correctly. OSPF LSA statistics and database counts of LSA types are properly exported. Recommendation to implement caching and retry mechanism in case of system failure. FR2: Gather FRR Routing Information → Participant Feedback: Need to clarify distinction between gathering and exporting in the FR. All OSPF types including type 7 are being gathered correctly. "Show running config" and system resources are correctly tracked. System resources should be validated and corrected if necessary. FR3: OSPF Route Anomaly Detection by TUI → Participant Feedback: Awaiting response from stakeholder regarding duplicate routes. Wrongly advertised routes not yet implemented. Need to add information page explaining anomaly types in the TUI. Additional page for root cause analysis is needed (examining RIB and FIB). TUI correctly shows when no anomalies are present. Currently only detecting predefined anomaly types. Title should be changed from "by TUI" to "FRR-MAD". FR4: Query and Display Routing Anomalies via TUI → Participant Feedback: Title should be updated to "FRR-MAD TUI". Anomalies are successfully displayed in the frontend. Need to add expected routes to AnomalyDetection to show "should" and "is" states in frontend. Logging capability needs to be implemented. FR14: FRRMon Replacement → Participant Feedback: Implementation appears complete, awaiting stakeholder's confirmation.

Continued on next page

Table B.1 – continued from previous page

Participant's Name	**Yannick Städeli, Mino Petrizzo, Roman Cvijanovic**
Overall Feedback	We conducted a 2-hour acceptance test session and found that much of the system is well implemented with only minor issues to correct. Specific observations include: • Question about double via route on 10.3.0.0/24 on router 103 • Suggestion to use "show ip ospf json" for a simpler and more general view on TUI • FIB should be added to the TUI (Yannick's suggestion) • Question about router anomaly on r103 (Roman's observation) Overall, the system shows good implementation progress with only minor adjustments needed.
Status	Done

B.1.2 Industry Partner

Table B.2: Acceptance Test Protocol - Julian Klaiber

Participant's Name	Julian Klaiber
Date	June 12, 2025
Location	Remote

Continued on next page

Table B.2 – continued from previous page

Participant's Name	Julian Klaiber
Tasks	FR1-1: Export OSPF Routing Metrics → Participant Feedback: Metrics are exported correctly. FR2: Gather FRR Routing Information → Participant Feedback: All the information needed is present and displayed correctly. FR3: OSPF Route Anomaly Detection by TUI → Participant Feedback: Anomaly detection is working correctly. Was tested on multiple devices with different configurations. No more false positives. FR4: Query and Display Routing Anomalies via TUI → Participant Feedback: TUI is understandable and setup is clear. Data export is working correctly to clipboard and file. Navigation is working as expected. Filtering function is very useful and is working very well. FR14: FRRMon Replacement → Participant Feedback: Metrics for the anomaly detection is working as expected. Logs could not be checked as there were no anomalies present at the time testing and before. FRR-MAD will be rolled out to the dev environment to test it thoroughly and later to production to check if it is matching the FRRMon implementation as expected.
Overall Feedback	Everything now seems to be working as intended. No further issues were noticed. Regarding FRRMon, we plan to roll out your tool to the development environment soon, let it run there for a few days, and then hopefully deploy it to production to verify if it behaves the same as FRRMon. Unfortunately, I couldn't find any anomalies in the development environment, so I wasn't able to test the log output.
Status	Done

Chapter C

Miscellaneous

C.1 Industry Partner's Anomaly Occurrence

When reconfiguring a specific interface, an unforeseen anomaly occurs. The goal is, that a prefix would be advertised as a stub network within a Router LSA (Type 1 LSA), rather than as an AS External LSA (Type 5 LSA). One can observe this change by comparing the initial configuration (Announce Prefix as Type 5) with the adjusted configuration (Announce Prefix as Type 1), specifically by examining the configuration section for `interface eth1`.

```
! Initial Configuration:
!
ip route 10.0.0.0/17 10.0.0.255
!
interface eth1
    ip address 10.0.0.0/27
exit
!
router ospf
    ospf router-id 100.100.100.100
    redistribute static route-map lanroutes metric-type 1
    redistribute connected route-map lanroutes metric-type 1
exit
!
```

Listing C.1: Announce Prefix as Type 5

```
! Adjusted Configuration:
!
ip route 10.0.0.0/17 10.0.0.255
!
```

```
interface eth1
    ip address 10.0.0.1/27
    ip ospf passive
    ip ospf area 0.0.0.0
exit
!
router ospf
    ospf router-id 100.100.100.100
    redistribute static route-map lanroutes metric-type 1
exit
!
```

Listing C.2: Announce Prefix as Type 1

To maintain clarity between the two configuration examples, the route-map is omitted here, but a possible version might look as follows:

```
!
access-list denyall seq 5 deny any
access-list localsite seq 10 permit 10.0.0.0/17
access-list localsite seq 15 permit 10.0.0.0/27
!
route-map lanroutes permit 10
    match ip address localsite
exit
!
```

Listing C.3: Example Access-List

However, while implementing this change, a specific error occurred. The following enumeration outlines the issue in detail:

1. config is changed according to code snippet above
2. stub network is announced correctly
3. Anomaly: static route `10.0.0.0/17` is missing
4. Solution Step 1: remove and add the access-list entry `10.0.0.0/17`
5. Anomaly: static route `10.0.0.0/27` is advertised and `10.0.0.0/17` still missing
6. Solution Step 2:
   (a) remove correct (`10.0.0.0/17`) network from access list
   (b) add wrong (`10.0.0.0/27`) network to access list
   (c) remove wrong (`10.0.0.0/27`) network from access list
   (d) add correct (`10.0.0.0/17`) network to access list

C.2 Development and Test Environment Architecture

AS described in the RFC 6996, private use ASNs were used [15].

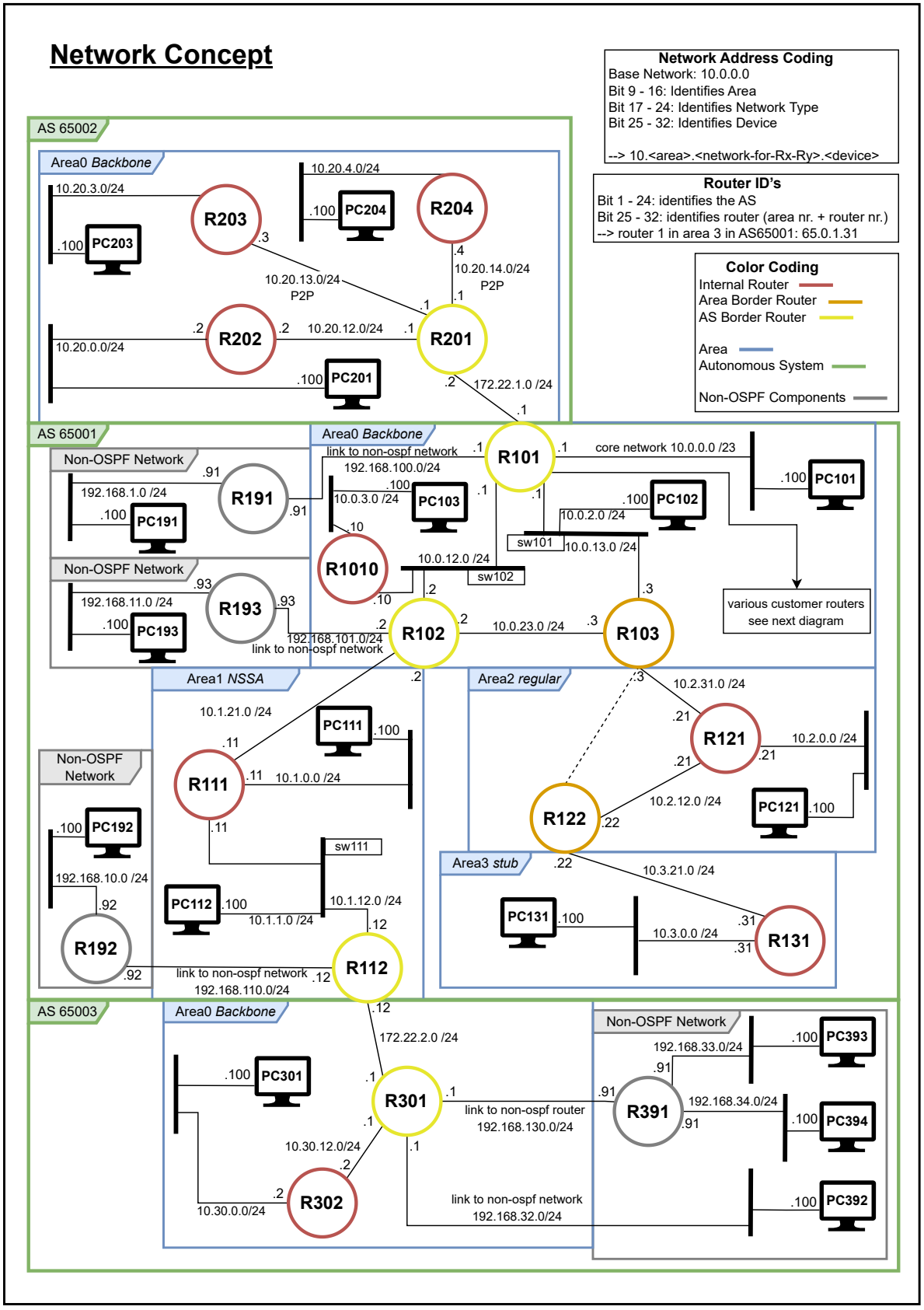


Figure C.1: Containerlab - Network Concept

C.3 Setup and Installation

In this subsection the FRR-MAD application installation and utilization will be outlined. Similar instructions are provided in GitHub README.

C.3.1 Build Instructions

Currently, no pre-built packages are available, requiring the project to be built before use. The applications should be built statically to eliminate dependency issues. The default version can be built by executing make, though this imposes restrictions on configuration path locations.

```
mkdir -p /tmp/frr-mad-binaries/
cd src/backend && CGO_ENABLED=0 GOOS=linux GOARCH=amd64 go build
    -ldflags='-s' -o /tmp/frr-mad-binaries/frr-mad-analyzer ./cmd/frr-analyzer
cd ../../
cd src/frontend && CGO_ENABLED=0 GOOS=linux GOARCH=amd64 go build
    -ldflags='-s' -o /tmp/frr-mad-binaries/frr-mad-tui ./cmd/tui
cd ../../
```

Listing C.4: Build Instructions

The default configuration path can be overridden during the build process using build flags. When the correct build flags (e.g., `-X configs.ConfigLocation=/path/to/configuration.yaml`) are specified, the built application reflects these changes. This is particularly valuable when the default configuration path does not conform to company policies or other requirements.

```
mkdir -p /tmp/frr-mad-binaries/
cd src/backend && CGO_ENABLED=0 GOOS=linux GOARCH=amd64 go build -ldflags='-s
    -X configs.ConfigLocation=/path/to/configuration.yaml' -o
    /tmp/frr-mad-binaries/frr-mad-analyzer ./cmd/frr-analyzer
cd ../../
cd src/frontend && CGO_ENABLED=0 GOOS=linux GOARCH=amd64 go build
    -ldflags='-s -X configs.ConfigLocation=/path/to/configuration.yaml' -o
    /tmp/frr-mad-binaries/frr-mad-tui ./cmd/tui
cd ../../
```

Listing C.5: Build Instructions with Build Flag Override

Once the application is built, a configuration file must be provided. An example configuration scheme is available here. All values are required, and if this application runs with restricted permissions, access to the specified locations and files must be granted.

```
default:
  tempfiles: /tmp/frr-mad
  exportpath: /tmp/frr-mad/exports
  logpath: /var/log/frr-mad
  # default is info
  debuglevel: error
```

```
7
8  frrmadtui:
9    pages:
10     ospf:
11       enabled: true
12     rib:
13       enabled: true
14     shell:
15       enabled: true
16
17 socket:
18   unixsocketlocation: /var/run/frr-mad
19   unixsocketname: analyzer.sock
20   sockettype: unix
21
22 aggregator:
23   frrconfigpath: /etc/frr/frr.conf
24   pollinterval: 5
25   socketpath: /var/run/frr
26
27 exporter:
28   # default: Port: 9091
29   OSPFRouterData: false
30   OSPFNetworkData: false
31   OSPFSummaryData: false
32   OSPFAsbrSummaryData: false
33   OSPFExternalData: false
34   OSPFNssaExternalData: false
35   OSPFDatabase: false
36   OSPFNeighbors: false
37   InterfaceList: false
38   RouteList: false
```

Listing C.6: Example Configuration File

C.3.2 Application Launch

After completing the build process, the build folder will contain two files:
`frr-mad-tui` and `frr-mad-analyzer`.
As previously explained, frr-mad-analyzer operates as a daemon, while frr-mad-tui functions as its client. The frr-mad-tui cannot operate without the frr-mad-analyzer daemon running. To start frr-mad-analyzer executing `frr-mad-analyzer start` is enough. This spawns a new child process that runs in the background. According to the configuration, a Unix Domain Socket will be created, different files touched and a network socket opened. The locations can be adjusted in said configuration.

For detailed information about the tool's capabilities, executing it without arguments displays a help menu.

```
$> frr-mad-analyzer
A CLI tool for managing the FRR-MAD application.

Usage:
  frr-mad-analyzer [command]

Available Commands:
  completion  Generate the autocompletion script for the specified shell
  help        Help about any command
  restart     Restart the FRR-MAD application
  start       Start the FRR-MAD application
  stop        Stop the FRR-MAD application
  version     show version number and exit

Flags:
  -h, --help   help for frr-mad-analyzer

Use "frr-mad-analyzer [command] --help" for more information about a command.
```

Listing C.7: frr-mad-analyzer help menu

Some commands provide extended options. While metric exports can be configured through the configuration file, the start command also accepts additional arguments. This is particularly useful during testing phases.

```
$> frr-mad-analyzer help start
Start the FRR-MAD application

Usage:
  frr-mad-analyzer start [flags]

Flags:
  -c, --configFile string    Provide path overwriting default configuration
      file location.
  -h, --help                 help for start
      --interface-list       Enable interface list metrics
      --ospf-asbr-summary    Enable OSPF ASBR summary metrics
      --ospf-database        Enable OSPF database metrics
      --ospf-external        Enable OSPF external route metrics
      --ospf-neighbors       Enable OSPF neighbor metrics
      --ospf-network         Enable OSPF network metrics
      --ospf-nssa-external   Enable OSPF NSSA external route metrics
      --ospf-router          Enable OSPF router metrics
      --ospf-summary         Enable OSPF summary metrics
      --route-list           Enable route list metrics
```

Listing C.8: frr-mad-analyzer command help

Once the daemon is running, starting frr-mad-tui is straightforward—simply execute the binary. Note that frr-mad-tui requires the configuration file location to be defined. Three methods are available:

- Place the configuration file at the default location: `/etc/frr-mad/main.yaml`
- Use build flags during the build process to specify an alternative configuration file location.
- Export an environment variable: `export FRR_MAD_CONFFILE=/path/to/configuration` with the configuration file path.