University of Zurich[UZH]

BACHELOR THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

Segment Routing
Optimizing Routing in Software-Defined Battlefield Networks

Samuele Giunta
Zurich, Switzerland
Student ID: 19-742-667

Supervisor: Rafael Hengen Ribeiro, Dr. Eryk Schiller
Date of Submission: August 22, 2022

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Bachelor Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Abstract

Tactical Battlefield Networks are networks used by the military to establish communication between units on the field. These networks pose diverse environmental challenges that need to be addressed. They are organized in so-called Mobile Ad Hoc Networks, that mimic a civilian cellphone network infrastructure. Testing performance in these networks proves more difficult than testing traditional fixed communication networks.

This thesis focuses on the development and implementation of an automated testing environment. This testing environment provides the user with the ability to convert arbitrary network topologies to a virtual environment. Further, traffic can be configured to automatically run on these networks and results can easily be fetched without the user directly having to interact with the virtual environment.

To assess the state of the implementation, first experiments with a small network topology were conducted and the results of these experiments was shown and discussed.

Zusammenfassung

Taktische Kampfnetzwerke sind Netzwerke, die das Militär für die Kommunikation zwischen Einheiten im Feld einsetzt. Diese Netzwerke stehen vor verschiedenen umweltbedingten Herausforderungen, die es zu bewältigen gilt. Sie sind in so genannten mobilen Adhoc-Netzwerken organisiert, die die Infrastruktur eines zivilen Mobilfunknetzes nachahmen. Die Prüfung der Leistung in diesen Netzen erweist sich als schwieriger als deren Prüfung in herkömmlichen statischen Kommunikationsnetzwerken.

Diese Arbeit konzentriert sich auf die Entwicklung und Implementierung einer automatisierten Testumgebung. Diese Testumgebung bietet dem Benutzer die Möglichkeit, beliebige Netztwerktopologien in eine virtuelle Umgebung zu konvertieren. Darüber hinaus kann der Datenverkehr so konfiguriert werden, dass er automatisisch das Netwerk testet. Die Ergebnisse können leicht abgerufen werden, ohne dass der Benutzer direkt mit der virtuellen Umgebung interagieren muss.

Um den Stand der Implementierung zu beurteilen, wurden erste Experimente mit einer kleinen Netzwerktopologie durchgeführt und die Ergebnisse dieser Experimente presäntiert und diskutiert.

Acknowledgements

I like to thank Dr. Eryk Schiller, Rafael Hengen Ribeiro, and the CSG Team for their constant support and all their helpful input. Furthermore, I like to thank my close friends and my brother, who always supported me during this thesis.

Contents

Chapter 1

Introduction

1.1 Motivation and Description of Work

Tactical Battlefield Networks (TBN) are networks that differ from traditional stationary networks. They are characterized by dynamic environmental conditions that constantly endangers nodes within them to suddenly and unexpectedly disconnect from the network. Research regarding TBNs is also usefull for civilian Mobile ad Hoc Networks (MANET). General insight into optimal usage of MANETs can further innovation in cellphone networks. A lot of network research focuses on stationary networks, that do not experience such harsh conditions.

Recent research has introduced the concept of Software Defined Networking (SDN) to MANETs. SDN redefines key networking principles and open new possibilities for future improvement of MANETs. A routing technique suited for SDN is Segment Routing (SR). SR is a stateless routing protocol and might improve performance in TBNs.

1.2 Problem Description

In order to successfully implement and test SDN and SR, an efficient test environment has to be implemented. The test environment has to be reliable and stable. Physical test environments are difficult and expensive to set up. Open-source tools exist to create and test virtual environments. The problem is to build an adequate system that can automate tedious processes, such as converting a network topology to the right format or getting traffic data to realistically load test the network.

This work build upon the Harmonia framework developed by the University of Zurich and aims to extend their emulated testing environment. This helps future researchers to conduct efficient tests for their SR algorithms.

1.3 Goals

The goal of this work is to implement and test an emulated testing environment. This testing environment should be easy to use, reliable and ease future contributions to the Harmonia project. Therefore, this thesis aims to reach the following goals:

- The implementation of an emulated test environment that is easily usable and reliable.
- The definition of useful metrics that can be used for tests.
- The conduction of first entry tests to ensure that the implemented test envrionment works as expected.

1.4 Report Outline

This thesis is structured the following way: Chapter 2 provides insight to the underlying technologies the test environment aims to support. Chaper 3 concerns itself with the practical implementation of the emulated test environment and also describes the used programming languages and frameworks. Chapter 4 provides insight into first experiments that were conducted using the newly implemented test environment. Finally, Chapter 5 summarizes the contribution of this thesis and discusses possible future work.

Chapter 2

Background and Related Work

This chapter describes the necessary background, the underlying principles, and the technologies required to understand the main issues and questions targeted in this thesis. Section 2.6 summarizes current approaches to test SDN, SR and TBNs. It leads over to the next chapter describing the implementation of an emulated testing environment.

2.1 Software Defined Networking

SDN is an established networking paradigm. SDN redefines key networking principles. A Network infrastructure that supports SDN decouples network control from forwarding and enables direct programmability of the network. [1]

2.1.1 Concept

Traditional IP networks, though widespread, do not adequately meet new demands needed for technologies such as cloud computing or Internet of Things. [2] Long-established IP Networks proof to be complex, face difficulties in expansibility and are hard to manage. [3]

SDN is a networking paradigm, proposed by Standford University, to change current limitations of traditional IP network infrastructures. SDN aims to seperate the control plane from the data plane. Thus network switches become simple forwarding devices and control logic is implemented in the SDN-controller. [2]

SDN can be defined as a hierarchical architecture, in which applications and high-level instructions reside in the top tier. The bottom tier consists of physical and virtual network switches, which - as already mentioned - act as simple forwarding devices. The SDN controller resides between the top and bottom tier of the architecture. Its purpose is to direct data traffic inside network planes, but also across different network planes. in an optimal way. The different network planes in hierarchical order from top to bottom are typically called the Application Plane, the Control Panel and the Data Plane.

Communication between the different planes is typically handled by North- and Southbound Interfaces. The Northbound Interface (NBI) is used for communication between the Control and Application Plane, whereas the Southbound Interface (SBI) is used for communication between the Control and Data Plane. [4]

Figure 2.1 visually shows the interaction between the different planes.

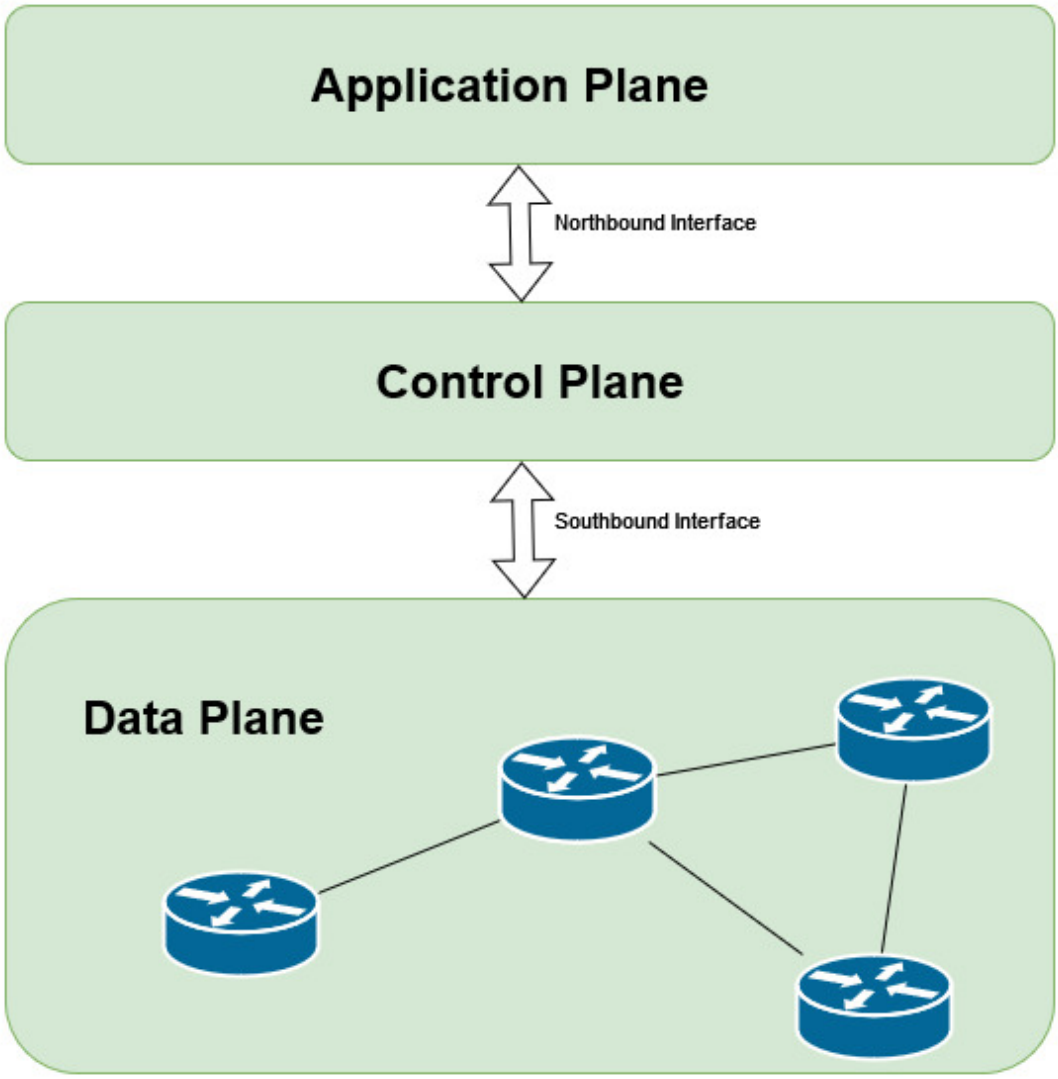


Figure 2.1: Hierarchical view of a SDN architecture

In Order to test SDN the Stanford University created OpenFlow 2.1.2. A communication protocol developed for researches that enables access to the Data Plane of a SDN network. [5]

Northbound Interface

The applications residing in the Application Plane communicate with the controller and have the ability to program the underlying infrastructure. This ability is generally granted

via an Interface. These so called NBIs have a variety of design goals to meet. Since these interfaces are used by applications to program the network, security and flexibility of NBIs are key aspects to be considered. [6]

Current research in this area is still occupied with finding approaches to construct intuitive and efficient NBIs. [7] Since NBIs aren't the main focus of this thesis, this small introduction to NBIs is needed and sufficient to understand the overall picture of SDN.

Southbound Interface

SBIs are the counterpart to NBIs. Whereas NBIs provide a communication interface between the application and control plane, SBIs provide an interface for the controller to communicate with the routers residing in the data plane. [8]

Different network protocols exist for this purpose. The most famous being *OpenFlow*. However, other protocols such as *NETCONF* or *RESTCONF* serve a similar purpose[9]. This thesis discusses the popular *OpenFlow* protocol in more detail. More information on *OpenFlow* can be found in the respective section *OpenFlow* 2.1.2.

Controller

SDN-controllers reside in the control plane and centralize the device administration and configuration of the network [10]. Implementations for the control plane generally favor multiple SDN-controllers spread across the network over one centralized controller. Distributed SDN-controllers preserve a logically centralized view of the network state while also mitigating process load and ensuring stability by removing a single point of failure from the network [11].

SDN-controllers play a vital role for measuring Quality of Service (QoS) metrics. Specifically, the throughput of a network relies on the performance of the different controllers. Thus, more often than not studies tend to test metrics such as *latency* [10] [12]. Further details for important QoS metrics used in this thesis can be found in the respective section 2.2.3.

Currently no industry standard for SDN-controllers exists [12]. Thus, research in this field is still needed and this theses provides a test environment to efficiently test new controllers.

2.1.2 OpenFlow

OpenFlow was first introduced by the Stanford University in 2008 as a way for researchers to run experimental protocols in networks. The committee also encouraged network vendors to add OpenFlow to their commercially available switches [5]. Openflow has come a long way since then. Nowadays OpenFlow is considered to be the most popular protocol standard for SDNs [13].

OpenFlow can be used as a programmable network protocol to remotely control forwarding tables in network routers and switches. OpenFlow enables access to state and statistics from the data plane and can remote issue routing commands regarding routing. [14]

[14, Fig. 2.2] illustrates the working of the OpenFlow Protocol.

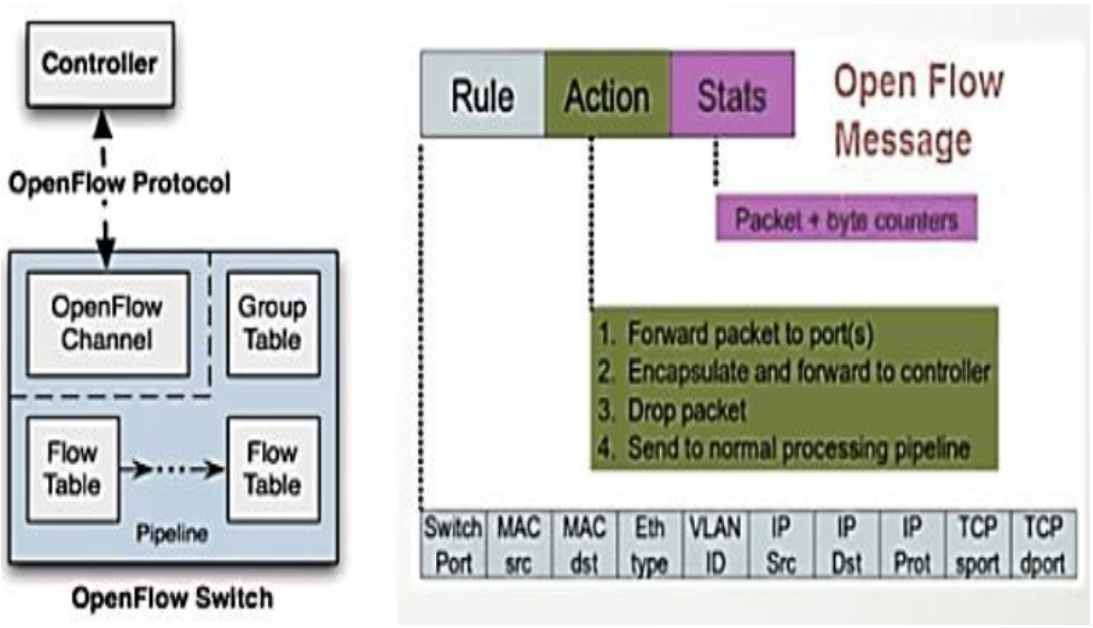


Figure 2.2: The OpenFlow protocol according to [14]

2.2 Tactical Battlefield Networks

TBN is a term to group networks who are primarily used by the military in field operations [15]. TBNs provide the ability for units to exchange data on the field and are thus a necessity in modern warfare. Furthermore, network requirements and capabilities are dependent on a variety of factors, such as terrain, required mobility, which specific communication sources are used and more [16].

2.2.1 Diverse Operational Environments

As already mentioned, TBNs operate under varying operational environments. This causes a multitude of challenges that have to be considered when studying TBNs. Sanchez, Evans and Minden [17] highlight the following challenges for TBNs:

- *Topology Management* is difficult in highly dynamic TBNs. There is a need for self-configuring mechanisms.
- *Mobile nodes* might be difficult to track in a large network.
- Computing *optimal routes* through nodes in a TBN is challenging.

Cheung and Yin [16] specify some of these aspects in a more concrete manner and introduce the following notions:

- *Limited Bandwidth and Latency* in TBNs. This results from the fact that highly mobile devices might be in use that can not generate much throughput for the network.
- *Intermittent Communication Links* in the network. Due to topological difficulties, networked notes might appear online and offline intermittently. Thus communication links are unstable and unreliable in TBNs.
- *Hidden Nodes and absence of Line of Sight (LOS)* can cause data collisions between nodes. [16, Fig. 2.3] illustrates this with an example.

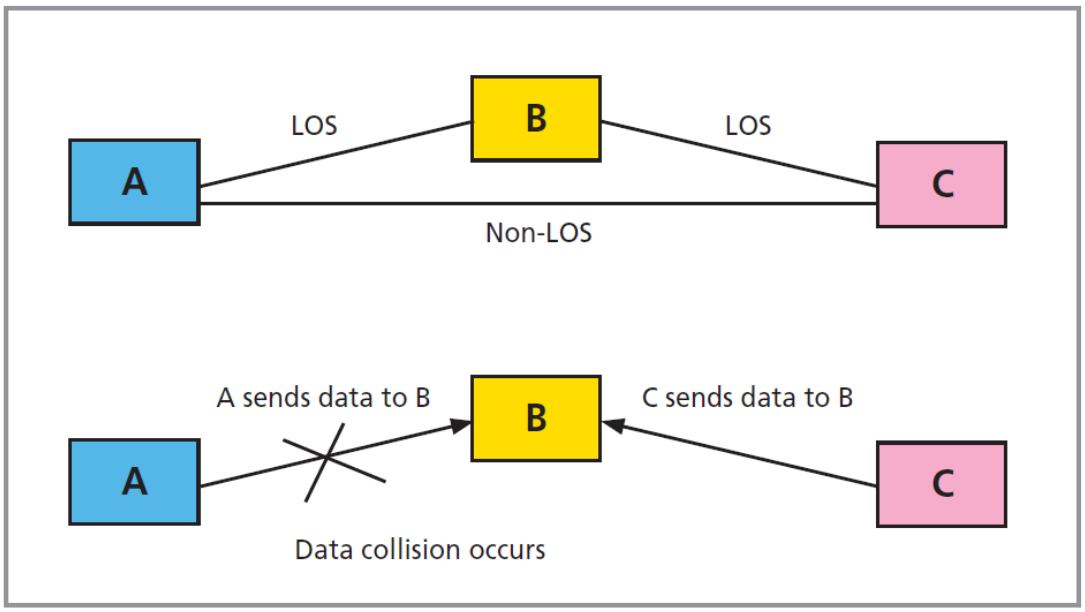


Figure 2.3: A Hidden Node scenario according to [16]

This thesis aims to provide new approaches to test solutions to these problems. The focus lies especially on the *Topological Management* and the computing of *optimal routes*.

2.2.2 Scalability

In TBNs each network entity is considered to be a network iteself. To be scalable, TBNs typically implement bidirectional data transfer between nodes[15]. In order to translate these notions into the real world, TBNs use MANETs to realize the networking architecture [16].

Figure 2.4 shows a high-level example of a TBN architecture. The green part highlights the focus of this thesis. Communication between different MANETs is not the primary goal of this thesis. Instead, communication within a MANET will be considered the primary case for the thesis.

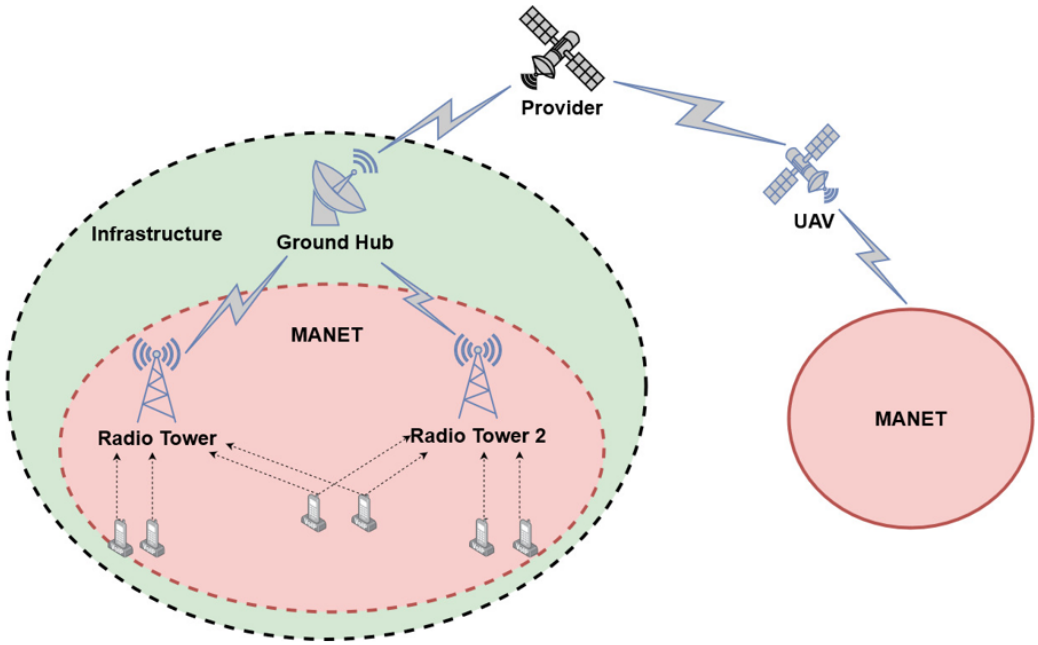


Figure 2.4: High-level example of a TBN architecture

Mobile Ad Hoc Networks

A MANET is defined by granting individual nodes in the network the ability to communicate with each other without passing base stations in the network [18]. In the spirit of bidirectional communication, nodes function both as transmitter and receiver of data. Further, MANETs typically contain a self-organising mechanism. If direct node-to-node communication is not possible, intermediate nodes are required to forward the data between the sender and the receiver [19].

2.2.3 Quality of Service

QoS refers to multiple characteristics of a network. QoS provides a framework to quantify the assurance of a network for an end user. This is done by choosing a measurement (e.g bandwidth) in the network and varying parameters. A requirement for the measurement has to be defined as well. If the requirement is fulfilled then we say that a system provides the measured assurance to an end user [20].

With the general term defined, the next step is to categorize important QoS measurements regarding TBNs.

QoS Measurements

Networks generally seek to optimize overall performance. This is done by measuring the overall *robustness* (connectivity), *efficiency* (throughput) and *speed of service*. However, for TBNs this approach might not yield the best results. TBNs require prioritization of

important messages, which in the traditional sense of QoS is often not given as a criteria [21].

Nevertheless, this thesis focuses mostly on bandwidth and throughput. The reason for this is that currently the implemented testing environment only supports static traffic. This thesis presents a first step to more accurate QoS measurements.

2.3 IPv6

In 1981 the University of Southern California first specified the Internet Protocol Version 4 (IPv4) [22]. The Internet Protocol Version 6 (IPv6) was first specified in 1995 as a successor to the IPv4 protocol [23].

This section concerns itself with the IPv6 addressing architecture and serves as a build-up for section 3.3.4.

IPv6 addresses are not assigned to nodes, but exclusively to their interfaces. Different types of IPv6 addresses exist, but for this thesis, the specific type of address does not concern us. IPv6 addresses follow a common structure *X:X:X:X:X:X:X:X*, where each *X* is a group that contains one to four hexadecimal digits. The length of an address is thus fixed to 128 bits, and leading zeroes may be omitted from the representation. In the case that one or multiple groups contain only zero values, they can be omitted by using the *"::"* string. As an example, the following two representations are considered equal [24]:

- ff01:0:0:0:0:0:0:101
- ff01::101

2.4 OSPF

Open Shortest Path First (OSPF) was first specified in 1989 by the Internet Engineering Task Force. The purpose of the Internet at that time was to serve as an academic and research network. A major part of Internet routing consisted of static routing. Systems that used dynamic routing used the Routing Information Protocol (RIP). With the rapid growth of the internet, the overhead of using RIP became unmanageable. Thus, the primary purpose of OSPF was to replace RIP with a more efficient routing protocol.

OSPF relies on link-state protocols, which are also called shortest-path first protocols. For the routing calculations, link-state protocols commonly use Dijkstra's shortest-path algorithm [25].

This thesis uses OSPF over IPv6, also called OSPF version 3 as the base protocol. SR functionality is tested against this protocol during experiments.

2.5 Segment Routing

SR is the primary routing algorithm tested in this thesis. This section aims to explain the underlying technology by explaining the overall concept and looking at two specific SR implementations.

2.5.1 Concept

Nodes that implement SR functionality use a list of segments to route packages through the network. The routing path of a packet is encoded in the segment list. Thus, routers in a SR network do not maintain state. Packet sources are defined as ingress nodes, whereas packet destinations are defined as egress nodes. For this to work, the encoding of segments onto a packet must be defined. This is done in the data plane. Further, the control plane specifies how and which segments are inserted into which header [26]. These two planes are described in the following subsections.

Data Plane

Segments are identified by a segment ID (SID). A SID can either have global significance or be significant to a local router depending on their type. Segments are typically categorized in three types:

- *Node SID*: this is the SID associated with the destination node in the topology. The goal of the router is to forward the packet towards this node on the shortest path.
- *Adjacency SID*: this is the SID associated with an interface. The goal of the router is to forward this packet over the specified link in the SID.
- *Service SID*: this type of SID effectively stops the routing. The router consumes the packet by delivering it to a predefined service.

Routers can perform different actions on segments. They can simply forward based on the active segment (CONTINUE). They can insert a segment and set that segment to active (PUSH). And Lastly, they can mark the next segment as an active segment (NEXT) [26].

Control Plane

The Control Plane defines how routers exchange informations about SIDs. The distribution of SIDs to different routers is currently supported by OSPF.

Routers have to be instructed to select a correct SR path. Different approaches exist to this problem. This thesis follows the SDN controller based approach. In this approach SDN controllers handle instructions regarding forwarding, inserting, and deleting segments [26].

2.5.2 SR-MPLS

SR can be directly implemented in the Multiprotocol Label Switching (MPLS) architecture without changing the hardware components of the MPLS forwarding plane. SR commands applying to SIDs can directly be mapped to MPLS commands [27].

2.5.3 SRv6

Segment Routing over IPv6 dataplane (SRv6) is the implementation used in the practical part of this thesis.

SRv6 specifies an SR implementation that works with IPv6 addresses. For this, a new IPv6 extension was proposed. This extension serves to encode the SR header in IPv6 packets [21].

2.6 Related Work

This section serves to provide an overview of different research papers that tested their contributions towards TBNs. Specifically, this section highlights the different implementations of testing environments and aims to provide a purpose for automated testing environments. It leads over to the next chapter, providing an implementation of an emulated test environment.

2.6.1 Existing Approaches towards testing TBNs

A multitude of approaches exist to improve the performance and QoS of TBNs. This thesis focuses on the current state regarding (i) the usage of SDNs to improve TBNs and (ii) the usage of SR to improve TBNs. This subsection provides a quick summary over existing approaches to test TBNs

First, Zhao et al. [28] provide a testbed for tactical scenario emulation in TBNs. Their testing environment implements a distributed SDN controller model and supports multiple traffic generators, including the Multi Generator Network Test Tool (MGEN). Their tests were carried out on a single topology that was manually crafted for their experiments.

Further, Flathagen and Benstuen [29] show that emulated testing environments can be used to approximate real-life conditions. They analyzed the control plane performance in a SDN TBN. Yet again, for the evaluation a manually crafted network topology was used and loaded with traffic.

Other studies implement small physical topologies running their SDN implementation, such as Phemius et al. [30]

2.6.2 Discussion

Test instances were mostly carried out on manually and specifically built network topologies. Emulated testing environments are used more frequently than physical testing environments. This makes sense, as emulated testing environments provide an easy alternative. It is feasible to omit physical testing environments, as Flathagen and Benstuen showed that emulated environments can sufficiently approximate the behaviour of physical environments.

In order to extend the current work, this thesis provides an automated testing environment. This environment can convert arbitrary topology data sets to a virtual environment. Traffic tests can then be carried out automatically, with the goal to ease the testing of TBNs, SDNs and SR algorithms.

Chapter 3

Emulated Test Environment Implementation

This Chapter describes the tools and frameworks used to implement an extensible testing environment. General approaches to convert topologies in the Graph Modelling Language (GML) format to suitable Graphical Network Simulator (GNS3) topologies are described. Furthermore, this chapter describes how traffic can be generated using MGEN. In this environment the already existing SR capabilities were tested against the traditional routing method OSPF. Section 3.2 describes the used tools and frameworks. The implemented Topology Converter is described in section 3.3. And finally, frameworks and considerations for load testing and traffic generation are described in section 3.4.

3.1 Existing Implementation

This section briefly describes the existing SR implementation this work aims to extend. 3.1.1 describes the underlying framework that enables data transfer between different entities in the network. Whereas section 3.1.2 describes the underlying implementation of the SR capabilities.

3.1.1 Harmonia FRR

This thesis extends and builds upon the HARMONIA project, a routing project created by the University of Zurich [31]. For traditional routing methods, the HARMONIA project relies on the FRRouting (FRR) project. FRR is an open-source routing suite that supports IPv6 and OSPF.

Furthermore, the HARMONIA project leverages its SR capabilities from ROSE. ROSE provides SRv6, an SR protocol designed to work with IPv6 addresses [32].

3.1.2 PCC and PCEC

The Path Computation Client (PCC) and Path Computation Element Controller (PCEC) are the major components providing SR functionalities.

The PCC uses SRv6 to insert and remove segments on a given node. The SRv6 functionality is provided by the ROSE SRv6 Node Manager [33]. Furthermore, it collects traffic statistics and reports them to the PCEC.

The PCEC receives traffic statistics from the nodes in the network. It provides SR policies to improve network traffic. To do this, the PCEC computes the link usage and tries to minimize congestion on the links.

3.2 Tools and Frameworks

This section aims to describe the relevant tools and frameworks that were used during the development of this thesis. More specifically, the used programming languages and third-party applications are discussed here.

3.2.1 Python

The Topology Converter was developed using Python. Python is a high-level general-purpose programming language [34]. Python allows the user to use multiple programming paradigms including object-oriented, functional-oriented, and aspect-oriented programming. Data types in Python can be both strongly and dynamically typed [35].

Choosing Python as the primary language for the Topology Converter has multiple reasons. Firstly, Python is an established tool in the scientific community that is used across multiple domains [36]. Secondly, Python is extensible and allows for easy use of external packages, that add programmable interfaces to existing applications [35]. Thirdly, the existing Harmonia implementation was already using Python as the main programming language. And lastly, Python possesses a worldwide open-source community that allows easy access to different libraries.

The newest Python version, Python3, was chosen for the implementation. Not only because it contains many useful built-in functions and many versatile open-source libraries, but also because official support for Python2 has ended on January 1st 2020 [37].

3.2.2 GNS3

The GNU Public Licence (GPL) governs the use of GNS3, an open-source graphical network simulator developed in Python that enables the emulation of complicated networks [38] [39]. GNS3 is used to emulate real-life networks without having the disadvantage of

implementing a testbed in hardware. Instead, GNS3 uses virtualization to its advantage to emulate and isolate each component in a network topology [39]. It is also important to mention a significant disadvantage this thesis faces with GNS3. A point-to-point Ethernet interface state on a typical physical network is reliant on the state of the other end. In such a physical network, both ends are in a down state if one end is disconnected or shut off. Compared to GNS3, if one end of a point-to-point Ethernet link shuts down in GNS3, the other end remains unaffected. Thus, fail-over scenarios on GNS3 are impossible to test by the conventional method of shutting down an interface [40].

3.2.3 Docker

Routers and Hosts function as the main components for the network emulation. These major components are virtualized and isolated using Docker [41].

Docker was initially launched as an open-source project in early 2013 [42]. Many approaches to virtualization exist, the most common among them being hypervisor-based virtualization. Contrary to hypervisor-based virtualization, which directly manages physical resources and allocates isolated slices of resources to its virtual instances [43], Docker instead uses a container-based virtualization approach. Containers work on the operating system level, creating protected portions of the operating system and making those available to the container instances [42].

Since Docker containers officially only support Linux-based operating systems, hypervisor-based and container-based approaches complement each other and allow development workflows on different operating systems. In fact, the recommended way of running Docker containers on a non-Linux machine is to install a hypervisor-based virtual instance of Linux and run further virtualization of applications in the virtual machine [42].

To summarize, Docker is both used to isolate major components in the emulation and to enable an easy development workflow for future extensions.

3.2.4 Multi-Generator Network Test Tool

MGEN is an open-source network test tool developed by the U.S Naval Research Laboratory. The tool enables users to generate real-time traffic patterns and thus provides multiple ways to load a network. Scripting MGEN allows for traffic to be sent dynamically across a network. Since MGEN version 5.0, protocols building on IPv6 are supported [44].

Supported Traffic Protocols

MGEN supports both the User Datagram Protocol (UDP) and Transmission Control Protocol (TCP) over IPv6. Each node in a network can start one MGEN process to send and receive packets. A MGEN process can perform multiple actions simultaneously. The process can receive incoming traffic by specifying a port, multiple ports, or a port range

to listen to. Simultaneously, a MGEN process can send traffic to destination ports on other nodes in the network [45].

Traffic Patterns

Traffic patterns describe a series of sequence-numbered messages that make up the traffic that MGEN generates. A controlled amount of network stress can be applied by varying the size and frequency of the messaging created by MGEN. With traffic patterns, MGEN may also emulate different network applications. Currently, MGEN version 5.0 supports four different types of patterns:

- *Periodic*: This pattern generates messages of a fixed size at a constant rate in seconds.
- *Poisson*: This pattern generates messages of a fixed size at statistically varying intervals at an average rate.
- *Burst*: Generates bursts of other MGEN pattern types with fixed message size. Bursts can occur uniformly distributed in time or exponentially distributed across a timeframe.
- *Jitter*: Generates messages of a fixed size at a rate in seconds. The actual rate is then randomly offset by a second jitter parameter. [45]

For this thesis, only the periodic pattern was used during testing, as explained in section 4.1

3.2.5 TRace Plot Realtime

The U.S Naval Research Laboratory also developed TRace Plot Realtime (TRPR). TRPR analyzes MGEN log files and creates summary statistics. TRPR can also be used to generate plots of the incoming traffic for a host [46].

TRPR is an open source software and its source code can be obtained on GitHub [47]. The plot files it generates can be displayed using Gnuplot [48].

Statistics gathered in the Evaluation chapter 4 were always gathered with the help of TRPR unless otherwise noted.

3.3 Topology Converter

The main goal of the implementation is to extend the current test environment. For this purpose, two distinct components that provide different functionalities were implemented.

This section concerns itself with the Topology Converter, whereas section 3.4 concerns itself with the traffic generator. Concerning the Topology Converter, the initial testing environment only supported one predetermined topology to test. Figure 3.1 shows this initial topology.

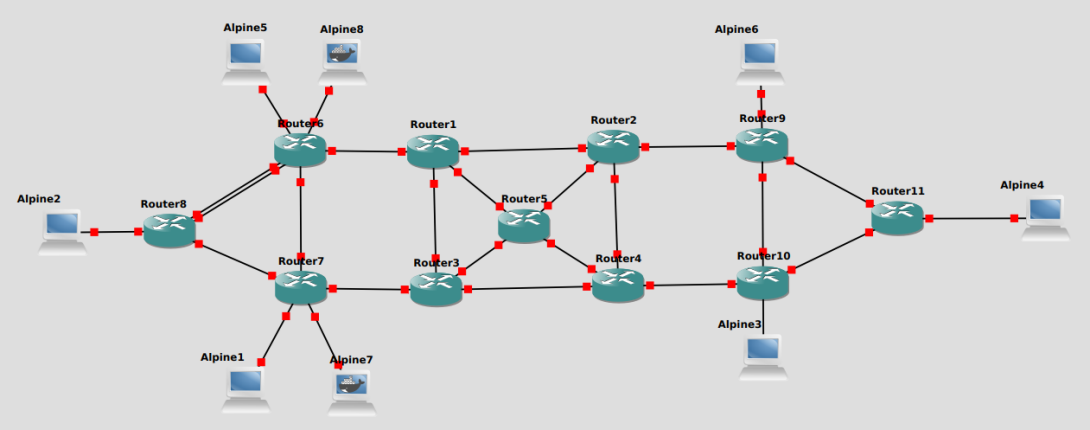


Figure 3.1: Initial topology available for testing

The Topology Converter enables users to convert arbitrary topologies provided in the GML data format to suitable GNS3 topologies. The following subsections describe important data formats, as well as the different components of the Topology Converter.

3.3.1 Graph Modelling Language (GML)

GML is a general-purpose file format used to model graphs. The University of Passau first proposed it in 2010 to serve as a portable file format for graphs. The structure of a GML file is hierarchical. The data structures used for defining graph attributes are lists containing key-value pairs. The hierarchical structure of the file enables users to precisely define nodes and edges of graphs. Users may define additional attributes, such as node names, as GML doesn't precisely define which key-value pairs can be used [49]. Figure 3.2 shows a simple GML file and a corresponding visualization of it.

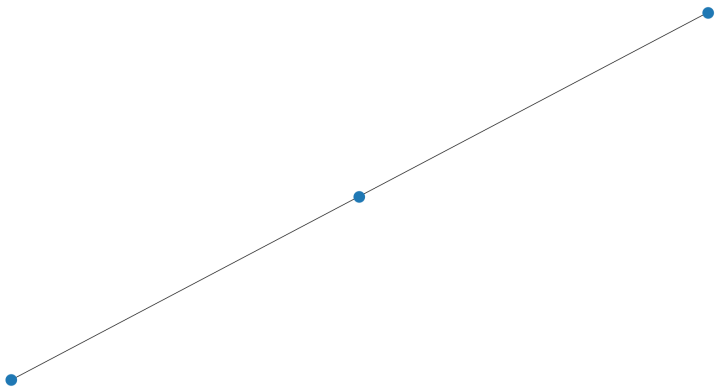
There are multiple reasons why GML was chosen as the primary file format supported by the Topology Converter. Firstly, graphs representing network topologies are easily created and obtained in the GML file format. Secondly, many scientific network graph repositories, such as Survivable fixed telecommunication Network Design library (SNDlib) 4.2.1, provide their problem instances in the GML format. Lastly, GML is supported in the open-source python library NetworkX [50], which eases the task of writing an appropriate conversion script for GML files.

3.3.2 .gns3 Files

GNS3 uses .gns3 files to store information about the topology of a GNS3 project. The .gns3 file is a structured JavaScript Object Notation (JSON) file that follows a hierarchical

```
graph [
    node [
        id A
    ]
    node [
        id B
    ]
    node [
        id C
    ]
    edge [
        source B
        target A
    ]
    edge [
        source C
        target A
    ]
]
```

(a) Code structure of a GML file



(b) The GML file visualized

Figure 3.2: A simple GML graph

order. In addition to specific information about the topology, the file also contains general properties related to the whole GNS3 project [51].

The following sections describe the structure of the different nodes and links needed for the successful generation of the .gns3 file.

Nodes

Routers and Hosts are tagged with the *nodes* key in the .gns3 file. Nodes have to be provided with a unique node id and a name. Furthermore, the number of available interfaces has to be specified. Since nodes are virtualized using Docker, the converter has to provide them with a unique container id. The respective Docker image has to be specified for both Routers and Hosts. The Docker image is also the main difference that arises between Routers and Hosts. Figure 3.3 shows a sample entry for a Router in a .gns3 file.

Links

Links are tagged with the *links* key in the .gns3 file. Links connect two nodes and provide a means of transport between them. A unique id has to be provided to each link. The nodes are provided via a list of node references, where the unique node id uniquely identifies a

```
{
    "compute_id": "local",
    "console_auto_start": false,
    "custom_adapters": [],
    "first_port_name": null,
    "height": 45,
    "label": {
        "rotation": 0,
        "style": "font-family: TypeWriter;
        font-size:10.0;
        font-weight: bold;
        fill: #000000;
        fill-opacity: 1.0;",
        "text": "ROUTER-3",
        "x": 0,
        "y": -20
        },
    "locked": false,
    "name": "ROUTER-3",
    "port_name_format": "Ethernet{0}",
    "port_segment_size": 0,
    "properties": {
        "adapters": 16,
        "aux": 5007,
        "console_http_path": "/",
        "console_http_port": 80,
        "console_resolution": "1024x768",
        "extra_volumes": [
            "/etc/frr"
            ],
        "start_command": null,
        "usage": ""
        },
    symbol": ":/symbols/classic/router.svg",
    "template_id": "12e1bc65-a14b-45f1-8d15-3c2b19066a9c",
    "width": 66,
    "x": 656,
    "y": -72,
    "z": 1
}
```


Figure 3.3: A sample Router contained in a .gns3 file

node in the topology. Furthermore, the interface on which the link connects to the node has to be specified. Figure 3.4 shows a sample link definition.

```
{
    "filters": {},
    "link_id": "eb229618-87bb-4590-ae24-2279390feb95",
    "link_style": {},
    "nodes": [
        {
            "adapter_number": 0,
            "label": {
                "rotation": 0,
                "style": "font-family: TypeWriter;
                font-size: 10.0;
                font-weight: bold;
                fill: #000000;
                fill-opacity: 1.0;",
                "text": "eth0",
                "x": 0,
                "y": -20
            },
            "node_id": "b725bcee-7bb5-45bc-89c0-27a06e954bc1",
            "port_number": 0
        },
        {
            "adapter_number": 0,
            "label": {
                "rotation": 0,
                "style": "font-family: TypeWriter;
                font-size: 10.0;
                font-weight: bold;
                fill: #000000;
                fill-opacity: 1.0;",
                "text": "eth0",
                "x": 0,
                "y": -20
            },
            "node_id": "4d1b284c-201c-4af5-80e2-56a88bfc6b9d",
            "port_number": 0
        }
    ],
    "suspend": false
}
```


Figure 3.4: A sample link connecting two nodes

3.3.3 Conversion from GML to GNS3

The conversion of a GML file to a GNS3 project takes place in two steps. First, the structure of the initial file must be mapped to a suitable data structure in the code. Then, the information contained within the GML file has to be translated to match the specifications of a GNS3 project. During translation, missing information has to either be provided beforehand by the user or generated by the script. The following sections describe this process in detail.

Configurable Variables

The user must enter additional information for the conversion to work. Basic information includes the path of a source GML file along with a destination path where the generated GNS3 project should be saved. Other variables allow the user to configure the behavior of the translator:

- *The template-file*: The file name of the GNS3 project template. It is contained within the project and represents the smallest possible .gns3 file (i.e., an empty topology). The generator copies information from this template that occurs in every .gns3 file. Figure 3.5 shows an empty topology that could be used as a template.
- *The number of transit interfaces*: An integer representing the number of transit interfaces generated per router. These interfaces are used during translation to connect different routers, hence the name transit interfaces.
- *The number of connected hosts*: An integer representing the maximum number of hosts that can potentially connect to a router. Increasing this number causes the converter to generate additional interfaces for the routers. These additional interfaces are exclusively used to attach hosts to the routers.

```
{
    "name": "untitled",
    "project_id": null,
    "revision": 9,
    "topology": {},
    "type": "topology",
    "version": "2.2.31"
}
```

Figure 3.5: A minimmal GNS3 project template

Loading the GML file

As previously mentioned, the Topology Converter has to map information in the GML file to a suitable data structure. For this purpose, the open-source python library NetworkX is

used to internally represent the graph during conversion. NetworkX fully supports GML files and provides basic operations such as reading and writing to GML files [50].

Once represented as a NetworkX graph, information within the GML file is easily accessible. This enables visualization and manipulation of the GML file. A sample GML file and its corresponding visualization with NetworkX are presented in Figure 3.2.

For the purpose of this thesis, each node in the graph functions as both receiver and sender in the network to mimic the behavior one might encounter in a MANET. To achieve this functionality within a GNS3 project, each node generates both a host and a router during translation. Section 3.3.3 describes this process in more detail.

Tranlating the GML file

A node defined in a GML file typically contains enough information to generate both a router and a host. The Topology Converter first translates information about the coordinates in the GML file to a host and a router. It then assigns them a generated UUID [52] to conform to GNS3 specifications. The translation also requires the converter to add a UUID for the Docker container. Furthermore, different Docker images are assigned to the router and host.

The Topology Converter then assigns interfaces to the router and host. The number of interfaces for a host is always capped to four. The amount of interfaces on the router corresponds to a user configured amount, where transit interfaces are dedicated for connections to other routers, and additional interfaces connect to hosts.

The GML file also defines the links between the routers. This information is stored in the NetworkX graph. The Topology Converter now loops over all the edges in the NetworkX graph and generates links accordingly. Additional links between host-router pairs are generated as well. All the links were provided with an upper throughput limit of 10 Megabits per second. The naming convention defines which interface will be assigned to which link, which is described in the next section 3.3.3.

Naming Conventions

The topology conversion and the FRR config generation require a consistent naming convention throughout the process. Specifically, router and host names must be consistent, and links have to attach to specific interfaces. The naming conventions are the following:

- Routers are named incrementally. Their name follows the structure *ROUTER-X*, where X is an integer. Routers are named in ascending order starting from one.
- Hosts are also named incrementally. Their name generally follows the structure *ALPINE-X-Y*. X is an integer that matches the connected router. Y is an integer used to enumerate the hosts. Hosts are incrementally enumerated within their respective router connection. Since hosts and routers always occur as pairs, the hostnames follow the structure *ALPINE-X-1*.

- Interfaces on routers and hosts follow the pattern *eth-X*, where X is an integer. Interfaces on a device are enumerated in ascending order and start at zero.
- Links between routers choose the next free transit interface on the source and destination.
- Links between routers and hosts follow a specific schema. The host always chooses its first free interface for a connection. The router selects the next free additional interface.

These conventions are later used to generate unique IPv6 addresses for the router and host interfaces. The specific details are described in the next section 3.3.4.

3.3.4 FRR Config Generation at Runtime

FRR routers record information about other connected devices in a config file named frr.conf. The file contains basic information, such as the hostname of the router. All generated interfaces are defined in the frr.conf file. Since this file changes depending on the specific device that is configured, it must be generated at runtime during the start of the Docker container.

For connections between two routers, no IPv6 address may be specified. The IPv6 address configured on the loopback interface of the router follows the pattern *fcff:X::1/32*, where X corresponds to the integer occurring in the router name converted to hexadecimal.

However, the host address deviates from the default configured IPv6 address. Thus, interfaces connected to a host have to define custom IPv6 addresses. The chosen format of the address is the following: *fd00:X:Y::1/96*, where X and Y correspond to the pattern *ALPINE-X-Y*. Hosts follow a similar pattern when assigning their address: *fd00:X:Y::2/96* and define a default route through *fd00:X:Y::1/96*. X and Y are converted to hexadecimal digits to conform IPv6 specifications.

3.4 Traffic Generator

The Traffic Generator is used to test different topologies. It does this by loading the network with traffic. Flows are defined to be specific traffic patterns sent from a source host to a destination host. This section describes the functionality of the Traffic Generator in detail.

3.4.1 MGEN vs. iPerf3

Earlier renditions of the Harmonia project used iPerf3 [53] for load testing the network.

Unfortunately, iPerf3 proved to be insufficient for the planned tests. The main flaw with iPerf3 is that each flow needs two separate iPerf3 instances to run. One instance must run on the source host to send the traffic, and one instance on the destination host to receive the traffic. This is not sustainable, stresses the network, and would have distorted test results due to the encountered performance bottleneck.

MGEN successfully reduces this bottleneck. MGEN can listen to multiple flows arriving at different ports with just one instance running. One MGEN instance is also capable of sending multiple flows at once. Furthermore, to both listen to and send flows MGEN only needs one running process per host. Lastly, TRPR eases the task of analyzing MGEN log files to gather data.

Thus, if we compare MGEN to iPerf3, it better suits the problem we want to test and does not distort data as much as iPerf3 would.

3.4.2 MGEN Script Files

MGEN script files enable a user to automate MGEN operations on running hosts. The most important commands are the following:

- *LISTEN*: orders MGEN to listen for flows on a specified port range. Received flows are logged to the console by default or logged to a log file.
- *ON*: orders MGEN to start a flow to a destination address with a specified port. A user can then specify the message size in bytes and a traffic pattern to use.
- *OFF*: orders MGEN to shut down a flow.

All these commands can be timed to execute after a defined number of seconds. This enables traffic to be sequentally started in the network [45].

Figure 3.6 shows a simple MGEN Script file. This file starts listening for incoming TCP traffic on port 5001. After 10 seconds, MGEN starts a TCP flow to a destination address sending two 1024 byte sized messages per second to the destination. Eventually, after 100 seconds of execution the flow is shut down.

3.4.3 Approach to Generate Script Files

The Traffic Generator generates an MGEN script file for each host in the topology. The traffic is either synthetically generated by a random distribution, or provided as a list of flows in a dedicated variable. The following subsections 3.4.3 and describe both approaches. For specific details regarding the synthesized traffic model, please refer to section 4.1.

```
# Start listening to tcp traffic
0.0 LISTEN TCP 5001

# Start sending a flow;
# Two messages per second of size 1024 bytes
10.0 ON 1 TCP DST 127.0.0.1/5000 PERIODIC [2 1024]

# Shut down the flow
100.0 OFF 1
```

Figure 3.6: A simple MGEN script file

Synthesized Traffic

Synthesized Traffic in this context refers to generated traffic that is not based on real-life data. It is an easy way to test different scenarios. The best approach is to choose a generation method that has proven to mimic real-life traffic sufficiently.

Given no Custom Traffic is present, the generator will always generate Synthesized Traffic. Each host will send a flow to each other host. Thus, if n hosts are present in the topology, approximately n^2 flows will be generated.

Custom Traffic

Custom Traffic describes traffic that was not generated, but rather gathered from empirical data. This is the preferred option between the two. The difficulty here arises in finding complete data sets for a topology.

3.4.4 Fetching Data from Hosts

A separate script is used to fetch statistics from the host. This script uses TRPR to analyze the MGEN log files. TRPR reports the received flows and the corresponding received rate over time in kilobits per second. Furthermore, TRPR generates a Gnuplot file to visualize the data. See Figure 3.7 for a sample plot. This data is then compared with the expected traffic rate to calculate the traffic loss that occurred in the network.

PCEC reports statistics about link utilization and the maximum link load in the network. With this link congestion in the network can be measured.

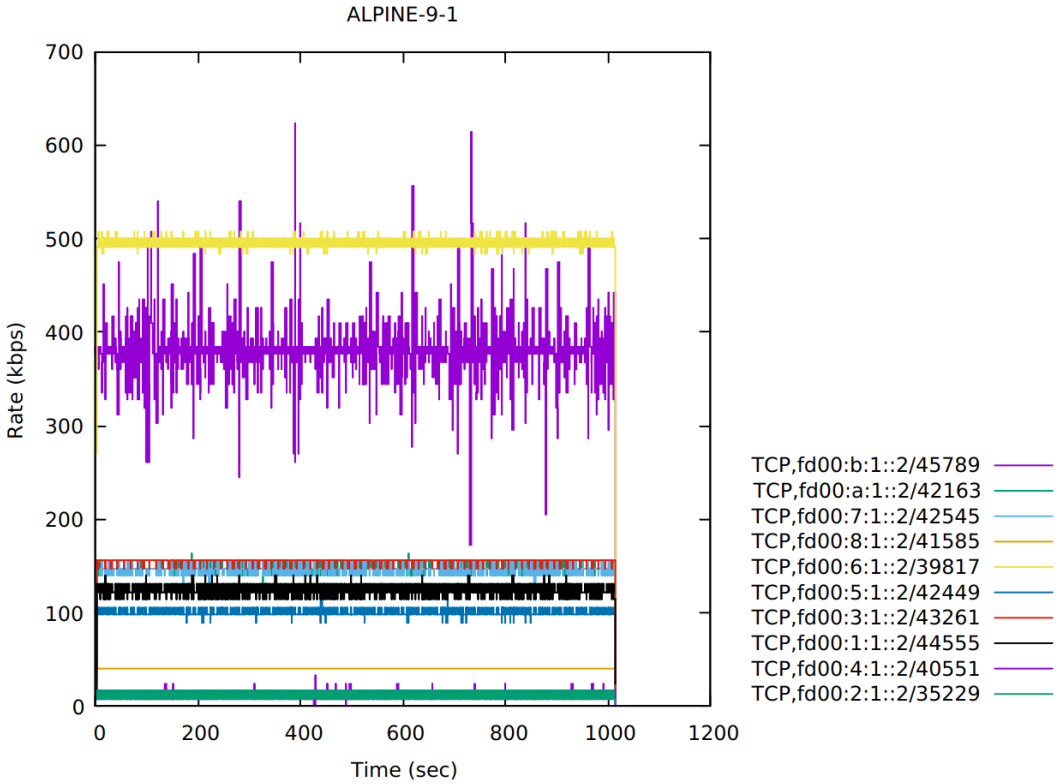


Figure 3.7: A sample Gnuplot that TRPR generated

Chapter 4

Evaluation

This Chapter describes the methodology and results of experiments that were conducted. The experiments aim to show (i) that the emulated test environment works as expected and (ii) that Harmonias SR algorithm is able to improve QoS in a network. The following sections describe the traffic model used for synthesized traffic 4.1, the network instance that was used for testing 4.2.2, and the experiment itself 4.3.

4.1 Traffic Model for synthesized traffic

The Traffic Model for the synthesized traffic follows the approach of the Random Gravity Model. This Model proved to closely approximate real-life traffic. The goal is to generate a traffic matrix T of size n x n where (i) n is the number of hosts in the topology and (ii) each entry *T(i,j)* represents the throughput of a flow sent from host i to host j.

The Random Gravity Model uses a random exponential distribution to distribute throughput between hosts. It depends on the total traffic t that passes through the network. Let p_i and p_o be equal to a probability obtained by a exponential distribution. More specifically, p_i defines the probability that traffic enters a node in the network and p_o defines the probability that traffic exits a node in the network. Now, mathematically the Random Gravity Model can be written as follows [54]:

$$T(i,j) = t * p_i(i)p_o(j) \tag{4.1}$$

Algorithm 1 shows the generation of the traffic matrix.

4.2 Problem Sets

Problem sets for TBNs are rare and the described characteristics are difficult to find in more common network topologies. Since the main focus of this thesis is to take the first

Algorithm 1: Synthetic Traffic Matrix Generation

input : The number of hosts in the topology n
The total traffic in the network t
output: A traffic matrix T

$t_i \leftarrow exp(n)$;
$t_o \leftarrow exp(n)$;
$p_i \leftarrow t_i / sum(t_i)$;
$p_o \leftarrow t_o / sum(t_o)$;
for $i = 0;\ i \leq n;\ i \leftarrow i + 1$ **do**
  for $j = 0;\ j \leq n;\ j \leftarrow j + 1$ **do**
    if $i \neq j$ **then**
      $T(i, j) \leftarrow t * p_i(i) * p_o(j)$
    end
  end
end

step towards automated network topology tests, common topologies are still used. Simple tests with common topologies pave the path for future tests with more sophisticated topologies. This section aims to describe the topology used for testing and provides sources for more test data.

4.2.1 SNDlib

SNDlib is a collection of test data used for fixed network design. Its primary goal is to provide the research community with realistic network design test instances. Furthermore, its test data acts as a common benchmark for testing, evaluating, and contrasting different network design models and algorithms [55].

The ABILENE topology is available on SNDlib and was used for testing the emulated test environment.

4.2.2 The ABILENE network topology

All of the experiments were carried out with the ABILENE network topology. This topology is a fixed telecommunication network representing a real-life network contained in the USA. [55, Fig. 4.1] shows the positions and links of the different nodes in the USA.

The ABILENE topology was chosen as a first test instance. Firstly, real traffic data is freely provided by the TOTEM project [56]. Further, other related work such as Tian et al. conducted similar tests to test a SR algorithm.

Figure 4.1: The ABILENE network topology pictured by [55]

4.3 Experiment with the ABILENE network topology

The first step was to convert the ABILENE topology from its GML form to a suitable GNS3 project. Once opened in GNS3, connectivity between the hosts was tested and ensured. The tests were conducted on a Personal Computer. The operating system was a virtualized instance of the Ubuntu operating system. the Ubuntu operating system had 16 Gigabyte of memory available and was assigned four CPU cores.

Figure 4.2 shows the converted ABILENE topology opened in GNS3.The following sections describe the two experiments that were conducted.

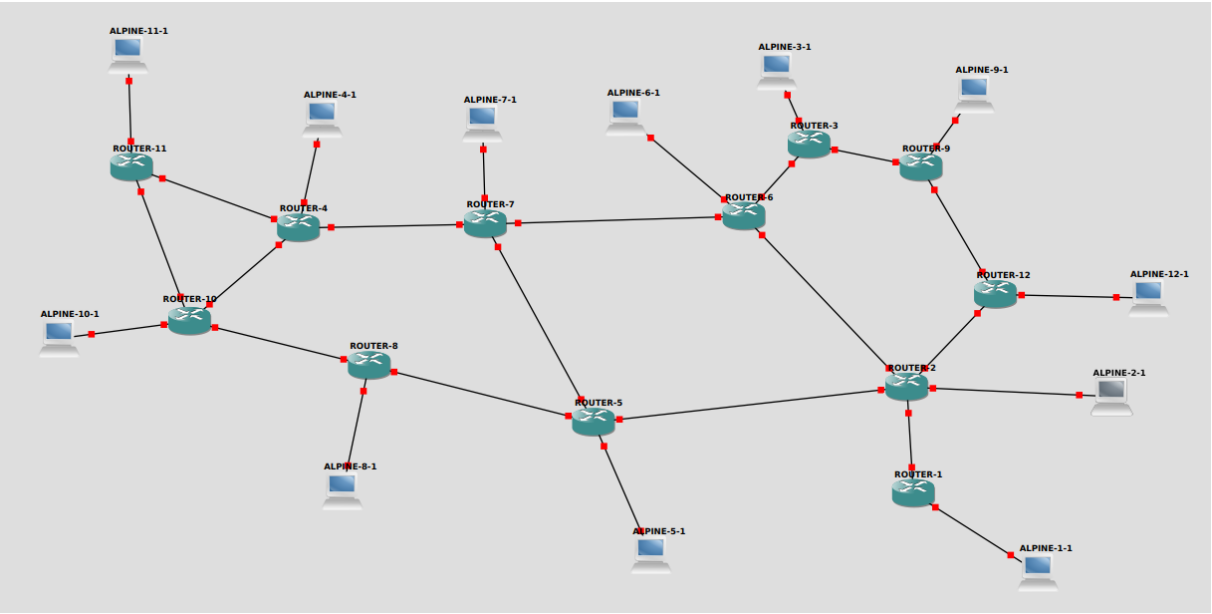


Figure 4.2: The ABILENE topology opened in GNS3

4.3.1 Custom Traffic

The first experiment was conducted with custom ABILENE traffic fetched from the TOTEM project [56]. The methodology and results are described here.

Methodology

First, the topology was started. This starts all the Docker container and generates the FRR configuration for each router and host. Then, the PCEC was started on *Router-5*, its output was logged to a log file.

Next, the custom traffic was started using the Traffic Generator 3.4. To stress the network and cause congestion, the values in the ABILENE traffic were interpreted as Bytes per second rather than bits per second. This was done to sufficiently stress the network for Harmonia to optimize the routing. Traffic was started sequentially, with each host waiting five seconds before starting a new flow. The test ran for 45 minutes. After the test duration the MGEN log files were analyzed and fetched by TRPR. Furthermore, the average link utilization and the maximum link utilization were reported via the PCEC.

This process had to be conducted two times. Once, with the Harmonia SR optimization disabled. A second time, with the Harmonia SR optimization enabled. The gathered data was compared to evaluate whether Harmonia SR optimization provided a benefit to the network.

Results

Figures 4.3, 4.4 and 4.5 show the gathered test results. The total traffic loss in both test cases was reported to be under one percent of the total traffic sent.

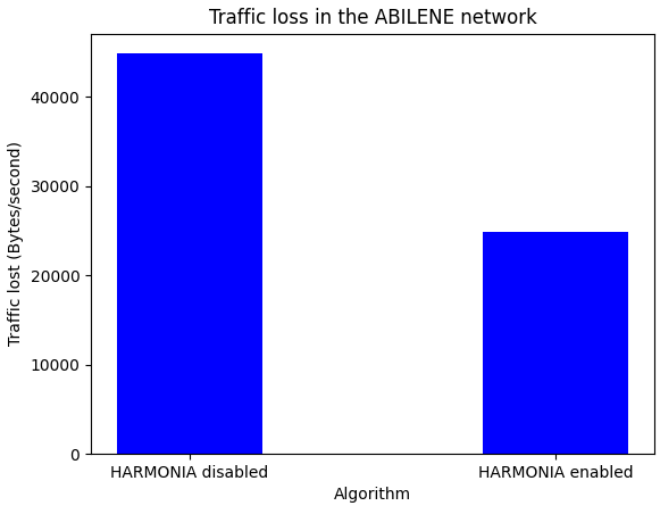


Figure 4.3: The aggregated traffic loss encountered in the network

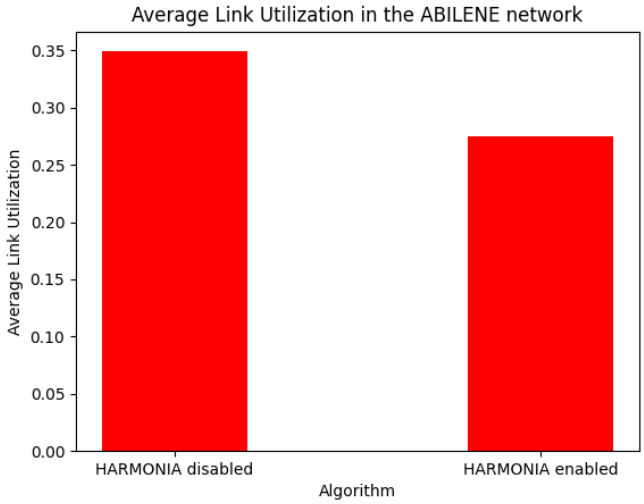


Figure 4.4: The reported average link utilization

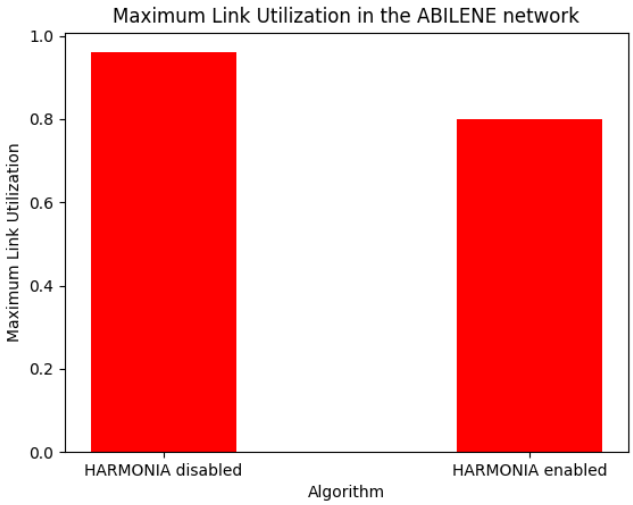


Figure 4.5: The reported maximum link utilization

Conclusion

The maximum link utilization never reached one, which lead to little congestion in the network. Nevertheless, Harmonias SR optimization was able to successfully reduce the average and maximum link utilization.

Since the measurements regarding traffic loss are so little, they are not significant and have to be disregarded.

Overall, in this test instance Harmonia successfully improved the QoS in the network.

4.3.2 Synthesized Traffic

The second experiment was conducted with synthesized traffic according to the Traffic Model 4.1. Its methodology and results are described here.

Methodology

The set up for the experiment essentially followed the same process as in experiment one. However, now synthesized traffic from the Traffic Model was used to stress the system. Four tests were conducted with increasing overall traffic in the network to limit test how much traffic the network can handle. The first test was conducted with an overall traffic rate of 1.000 Kilobytes per second, the second test with an overall rate of 10.000 Kilobytes per second, the third with an overall rate of 100.000 Kilobytes per second and the last test with an overall rate of 1.000.000 Kilobytes per second. Test runs were kept shorter for these tests, with the total test duration only being 15 minutes.

These test runs were also conducted two times. Once with Harmonia SR optimization enabled, and once with disabled optimization.

Results

Tables 4.1 and 4.2 show the lost traffic compared to the overall send traffic in percent. The numeric traffic loss can be found in figure 4.6.

The average and maximum link utilization can be found in figures 4.7 and 4.8 respectively.

Table 4.1: Traffic loss without Harmonia optimization enabled

Kilobytes per second	**Traffic loss in percent**
1.000	0.098%
10.000	23.8%
100.000	81.96%
1.000.000	97.7%

Table 4.2: Traffic loss with Harmonia optimization enabled

Kilobytes per second	**Traffic loss in percent**
1.000	0.076%
10.000	42.97%
100.000	48.14%
1.000.000	98.7%

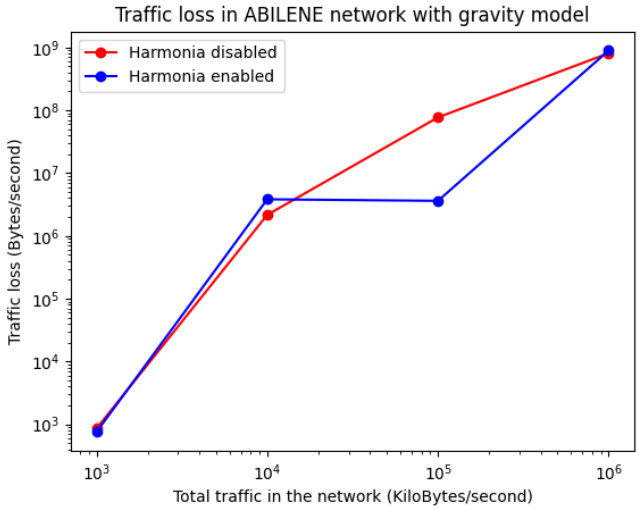


Figure 4.6: The reported traffic loss for synthesized traffic

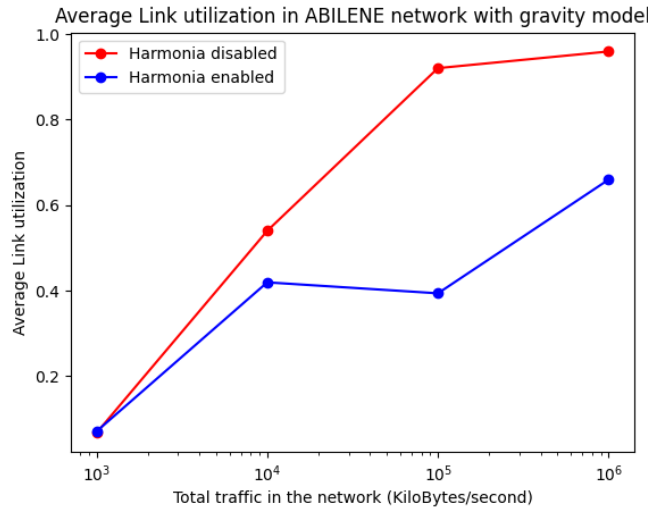


Figure 4.7: The reported average link utilization with synthesized traffic

Conclusion

These test runs show that the limits of the Harmonia SR optimization. However, the runs conducted with traffic over 100.000 Kilobytes per second should be disregarded. There is a high risk that the underlying virtual machine was overloaded and thus test results may have been distorted. There is also the problem that such high traffic volumes overload the links, which are capped to only 10 Megabits per second.

For runs with lower frequencies, Harmonia successfully managed to reduce the average link cost. Though, in this test instance no significant change of the maximum link utilization can be seen.

Conclusively, the data provided by experiment one has a higher quality than the one provided here. Although, this test instance shows that further tests are needed to sufficiently

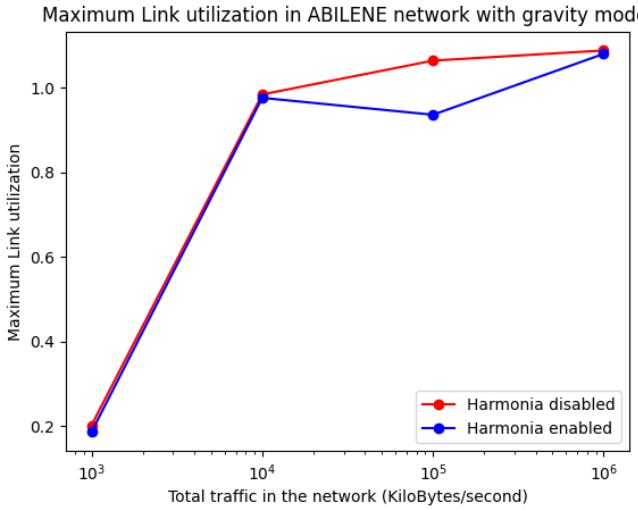


Figure 4.8: The reported maximum link utilization with synthesized traffic

validate the test results.

4.3.3 Further Topologies suited for future Tests

For future tests, all of the topologies used by Tian et al. would be useful to test [57]. Their test instances include topologies that contain more nodes than the ABILENE topology. These topologies were too big to test during this thesis, as a personal computer does not have enough ressources to successfully test them. Further, the smaller topologies could be tested as well to validate the results from experiment one. This was not done here due to time constraints.

Chapter 5

Summary and Conclusion

TBNs provide a diverse set of challenges to overcome. SDNs change major networking paradigms and might provide improvements for MANETs. Furthermore, a SDN controller can report the overall efficiency of a network. SR provides efficient routing algorithms and might synergize with SDNs.

This thesis investigated possibilities to automate tests for these technologies. It did this by investigating related work and summarizing their used test environments. This showed that most papers conducting tests on TBNs use custom built topologies and manually configure and test them. Further, this thesis produced two products. The first one being a fully implemented Topology Converter that can convert arbitrary topologies in the GML format to GNS3 topologies. The second one being a Traffic Generator that can be used to stress test these networks.

To asses the state of the implementation, multiple tests with the ABILENE topology were conducted. These tests mostly show that the testing environment works and provide some insights into the efficiency of the Harmonia SR optimization algorithm.

5.1 Future Improvements

The current testing environment works well to test fixed telecommunication networks, such as problem instances from SNDlib. The networks can be sufficiently loaded to observe improvements when enabling the Harmonia SR optimization algorithm. Specifically, test results show that Harmonias SR optimization can successfully lower average link utilization.

However, the current test environment does not yet successfully emulate the conditions TBNs experience. Further, the generation of synthetic traffic only allows one traffic model at the moment. The Traffic Generation itself only provides static traffic, which does not mimic real-life scenarios encountered in TBNs. The testing environment will have to be updated once Harmonia FRR can handle dynamic traffic. Also, since TBNs topology data is rare, a dataset for testing might be created specifically for them.

The experiments that were conducted show results, but these are ultimately not verifiable on their own. Further tests would be needed to obtain sound results regarding the Harmonia SR optimization. A further step would be to test topologies that contain more nodes and edges. For this, more processing power would be needed as to not distort potential results.

Bibliography

[1] D. King, C. Rotsos, A. Aguado, N. Georgalas, and V. Lopez, "The software defined transport network: Fundamentals, findings and futures," in *2016 18th International Conference on Transparent Optical Networks (ICTON)*, 2016, pp. 1–4.

[2] L. Xingtao, G. Yantao, W. Wei, Z. Sanyou, and L. Jiliang, "Network virtualization by using software-defined networking controller based docker," in *2016 IEEE Information Technology, Networking, Electronic and Automation Control Conference*, 2016, pp. 1112–1115.

[3] T. Benson, A. Akella, and D. Maltz, "Unraveling the complexity of network management," in *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*, ser. NSDI'09. USA: USENIX Association, 2009, p. 335–348.

[4] K. Kirkpatrick, "Software-defined networking," *Commun. ACM*, vol. 56, no. 9, p. 16–19, sep 2013. [Online]. Available: https://doi.org/10.1145/2500468.2500473

[5] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, p. 69–74, mar 2008. [Online]. Available: https://doi.org/10.1145/1355734.1355746

[6] C. Banse and S. Rangarajan, "A secure northbound interface for sdn applications," in *2015 IEEE Trustcom/BigDataSE/ISPA*, vol. 1, 2015, pp. 834–839.

[7] S. Alalmaei, Y. Elkhatib, M. Bezahaf, M. Broadbent, and N. Race, "Sdn heading north: Towards a declarative intent-based northbound interface," in *2020 16th International Conference on Network and Service Management (CNSM)*, 2020, pp. 1–5.

[8] P. L. Ventre, M. M. Tajiki, S. Salsano, and C. Filsfils, "Sdn architecture and southbound apis for ipv6 segment routing enabled wide area networks," *IEEE Transactions on Network and Service Management*, vol. 15, no. 4, pp. 1378–1392, 2018.

[9] M. Garrich Alabarce and A. Bravalheri, "Overview of south-bound interfaces for software-defined optical networks," in *2018 20th International Conference on Transparent Optical Networks (ICTON)*, 2018, pp. 1–5.

[10] A. A. M. Alraawi and S. A. N. Adam, "Performance evaluation of controller based sdn network over non-controller based network in data center network," in *2020 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE)*, 2021, pp. 1–4.

[11] T. Zhang, A. Bianco, and P. Giaccone, "The role of inter-controller traffic in sdn controllers placement," in *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2016, pp. 87–92.

[12] Y. Zhao, L. Iannone, and M. Riguidel, "On the performance of sdn controllers: A reality check," in *2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*, 2015, pp. 79–85.

[13] F. Hu, Q. Hao, and K. Bao, "A survey on software-defined network and openflow: From concept to implementation," *IEEE Communications Surveys Tutorials*, vol. 16, no. 4, pp. 2181–2206, 2014.

[14] F. M. Shuker, "Improvement of routing protocol for iot network using sdn/openflow," in *2019 1st AL-Noor International Conference for Science and Technology (NICST)*, 2019, pp. 54–57.

[15] J. Nobre, D. Rosario, C. Both, E. Cerqueira, and M. Gerla, "Toward software-defined battlefield networking," *IEEE Communications Magazine*, vol. 54, no. 10, pp. 152–157, 2016.

[16] L. Cheung and C. Yin, "Designing tactical networks – perspectives from a practitioner," pp. 122–132, 2011. [Online]. Available: https://www.dsta.gov.sg/docs/default-source/dsta-about/designing-tactical-networks-perspectives-from-a-practitioner.pdf?sfvrsn=2

[17] R. Sanchez, J. Evans, and G. Minden, "Networking on the battlefield: challenges in highly dynamic multi-hop wireless networks," in *MILCOM 1999. IEEE Military Communications. Conference Proceedings (Cat. No.99CH36341)*, vol. 2, 1999, pp. 751–755 vol.2.

[18] W. Tang and W. Guo, "A path reliable routing protocol in mobile ad hoc networks," in *2008 The 4th International Conference on Mobile Ad-hoc and Sensor Networks*, 2008, pp. 203–207.

[19] B. Lan, "Routing protocol design based on mobile ad hoc network," in *2011 Third International Conference on Multimedia Information Networking and Security*, 2011, pp. 176–178.

[20] Z. Rahaman and A. Das, "An algorithm to enhance the quality of service in mobile adhoc network," in *2012 2nd IEEE International Conference on Parallel, Distributed and Grid Computing*, 2012, pp. 355–358.

[21] G. L. Mayhew, "Quality of service in mission orientated ad-hoc networks," in *2007 IEEE Aerospace Conference*, 2007, pp. 1–9.

[22] J. Postel, "Internet protocol," Internet Requests for Comments, RFC Editor, RFC 791, December 1998. [Online]. Available: https://www.rfc-editor.org/rfc/rfc791.txt

[23] S. Deering and R. Hinden, "Internet protocol, version 6 (ipv6) specification," Internet Requests for Comments, RFC Editor, RFC 2460, December 1998. [Online]. Available: https://www.rfc-editor.org/rfc/rfc2460.txt

[24] S. Deering and R. Hinden, "Ip version 6 addressing architecture," Internet Requests for Comments, RFC Editor, RFC 2460, February 2006. [Online]. Available: https://www.rfc-editor.org/rfc/rfc4291.txt

[25] J. T. Moy, *OSPF : anatomy of an Internet routing protocol*, 1st ed. Reading, Mass: Addison-Wesley, 1998.

[26] C. Filsfils, N. K. Nainar, C. Pignataro, J. C. Cardona, and P. Francois, "The segment routing architecture," in *2015 IEEE Global Communications Conference (GLOBECOM)*, 2015, pp. 1–6.

[27] C. Filsfils, P. Camarillo, J. Leddy, D. Voyer, S. Matsushima, and Z. Li, "Segment routing with the mpls data plane," Internet Requests for Comments, RFC Editor, RFC 8986, February 2021. [Online]. Available: https://www.rfc-editor.org/rfc/rfc8986.txt

[28] Q. Zhao, A. J. Brown, J. H. Kim, and M. Gerla, "An integrated software-defined battlefield network testbed for tactical scenario emulation," in *MILCOM 2019 - 2019 IEEE Military Communications Conference (MILCOM)*, 2019, pp. 373–378.

[29] J. Flathagen and O. I. Bentstuen, "Control plane performance in tactical software defined networks," in *MILCOM 2018 - 2018 IEEE Military Communications Conference (MILCOM)*, 2018, pp. 1001–1006.

[30] K. Phemius, J. Seddar, M. Bouet, H. Khalifé, and V. Conan, "Bringing sdn to the edge of tactical networks," in *MILCOM 2016 - 2016 IEEE Military Communications Conference*, 2016, pp. 1047–1052.

[31] R. Ribeiro, F. Marino, M. Buck, E. Schiller, and C. Feng, "Reasearch and development of a software element in support of routing optimizations under certain conditions (harmonia)," 2022.

[32] "Frrouting project." [Online]. Available: https://frrouting.org/

[33] Rose, "Research on open srv6 ecosystem." [Online]. Available: https://netgroup.github.io/rose/

[34] Dave Kuhlman, "A Python Book: Beginning Python, Advanced Python, and Python Exercises," 4 2012. [Online]. Available: https://web.archive.org/web/20120623165941/http://cutter.rexx.com/~dkuhlman/python_book_01.html

[35] Chris M, "BeginnersGuide/Overview," 9 2019. [Online]. Available: https://wiki.python.org/moin/BeginnersGuide/Overview

[36] K. J. Millman and M. Aivazis, "Python for scientists and engineers," *Computing in Science & Engineering*, vol. 13, no. 2, pp. 9–12, 2011.

[37] Python.org, "Sunsetting Python 2." [Online]. Available: https://www.python.org/doc/sunset-python-2/

[38] "Gns3 network simulator software." [Online]. Available: https://gns3.com/software

[39] L. N. Dayanand, B. Ghorbani, and S. Vaghri, "A survey on the use of gns3 for virtualizing computer networks," 2016.

[40] C. Welsh, "Gns3 network simulation guide," in *GNS3 Network simulation guide*, ser. Community experience distilled. Birmingham: Packt Publishing, 2013, p. 131.

[41] Docker.com, "Docker: Build ship and run any app anywhere." [Online]. Available: https://www.docker.com/

[42] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux journal*, vol. 2014, no. 239, p. 2, 2014.

[43] Z. Li, M. Kihl, Q. Lu, and J. A. Andersson, "Performance overhead comparison between hypervisor and container based virtualization," in *2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)*, 2017, pp. 955–962.

[44] nrl.navy.mil, "Multi-generator (mgen) network test tool." [Online]. Available: https://www.nrl.navy.mil/Our-Work/Areas-of-Research/Information-Technology/NCS/MGEN/

[45] nrl.navy.mil, "Mgen user's and reference guide." [Online]. Available: https://github.com/USNavalResearchLaboratory/mgen/blob/master/doc/mgen.pdf

[46] nrl.navy.mil, "Trpr user's guide version 2.1b5." [Online]. Available: https://github.com/USNavalResearchLaboratory/trpr/blob/master/trpr.pdf

[47] nrl.navy.mil, "Trace plot realtime (trpr) distribution (2.1b10)," https://github.com/USNavalResearchLaboratory/trpr, 2020.

[48] H.-B. Bröker, J. Campbell, R. Cunningham, D. Denholm, G. Elber, R. Fearick, C. Grammes, L. Hart, L. Hecking, P. Juhász, T. Koenig, D. Kotz, E. Kubaitis, R. Lang, T. Lecomte, A. Lehmann, A. Mai, B. Märkisch, E. A. Merrit, P. Mikulík, C. Steger, S. Takeno, T. Tkacik, J. Van der Woude, J. R. Van Zandt, A. Woo, and J. Zellner, "Gnuplot 4.6: an interactive plotting program," http://gnuplot.sourceforge.net/, April 2013.

[49] M. Himsolt, "Gml: A portable graph file format," 2010.

[50] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using networkx," in *Proceedings of the 7th Python in Science Conference*, G. Varoquaux, T. Vaught, and J. Millman, Eds., Pasadena, CA USA, 2008, pp. 11 – 15.

[51] api.gns3.net, "The gns3 files." [Online]. Available: http://api.gns3.net/en/latest/file_format.html

[52] P. Leach, M. Mealling, and R. Salz, "A universally unique identifier (uuid) urn namespace," Internet Requests for Comments, RFC Editor, RFC 4122, July 2005. [Online]. Available: https://www.rfc-editor.org/rfc/rfc4122.txt

[53] J. Dugan, S. Elliott, B. A. Mah, J. Poskanzer, and K. Prabhu, "iperf3, tool for active measurements of the maximum achievable bandwidth on ip networks," 2014. [Online]. Available: https://github.com/esnet/iperf

[54] M. Roughan, "Simplifying the synthesis of internet traffic matrices," *SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 5, p. 93–96, oct 2005. [Online]. Available: https://doi.org/10.1145/1096536.1096551

[55] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly, "SNDlib 1.0–Survivable Network Design Library," in *Proceedings of the 3rd International Network Optimization Conference (INOC 2007), Spa, Belgium*, April 2007, http://sndlib.zib.de, extended version accepted in Networks, 2009. [Online]. Available: http://www.zib.de/orlowski/Paper/OrlowskiPioroTomaszewskiWessaely2007-SNDlib-INOC.pdf.gz

[56] S. Uhlig, B. Quoitin, J. Lepropre, and S. Balon, "Providing public intradomain traffic matrices to the research community," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 1, p. 83–86, jan 2006. [Online]. Available: https://doi.org/10.1145/1111322.1111341

[57] Y. Tian, Z. Wang, X. Yin, X. Shi, Y. Guo, H. Geng, and J. Yang, "Traffic engineering in partially deployed segment routing over ipv6 network with deep reinforcement learning," *IEEE/ACM Trans. Netw.*, vol. 28, no. 4, p. 1573–1586, aug 2020. [Online]. Available: https://doi.org/10.1109/TNET.2020.2987866

Abbreviations

FRR FRRouting

GML Graph Modelling Language

GNS3 Graphical Network Simulator

GPL GNU Public Licence

IPv4 Internet Protocol Version 4

IPv6 Internet Protocol Version 6

JSON JavaScript Object Notation

LOS Line of Sight

MANET Mobile ad Hoc Networks

MGEN Multi Generator Network Test Tool

MPLS Multiprotocol Label Switching

NBI Northbound Interface

OSPF Open Shortest Path First

PCC Path Computation Client

PCEC Path Computation Element Controller

QoS Quality of Service

RIP Routing Information Protocol

SBI Southbound Interface

SDN Software Defined Networking

SID segment ID

SNDlib Survivable fixed telecommunication Network Design library

SR Segment Routing

SRv6 Segment Routing over IPv6 dataplane

TBN Tactical Battlefield Networks

TCP Transmission Control Protocol

TRPR TRace Plot Realtime

UDP User Datagram Protocol

List of Figures

List of Tables

List of Algorithms

Appendix A

Emulated Testing Environment source code

The source code for the emulated testing environment is not publicly available. It contains confidential information that only members of the Communication Systems Group are able to access. The source code was handed in with the thesis and contains documentation on how to use the Topology Converter and the Traffic Generator.

Appendix B

Additional Files

The following list of files is available under the following **link**.

1. Bachelor thesis as pdf (optimizing-routing-in-software-define-battlefield-networks.pdf)
2. The abilene dataset as a GML file (abilene.gml)
3. The abilene dataset as a GNS3 project (abilene.gns3)
4. The gathered test data and plots used in the evaluation section (Data-and-plots.zip)