UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

SOFTWARE DEFINED NETWORKS OVER CHILEAN WIDE AREA NETWORKS: BUILDING RESILIENCE AGAINST NATURAL DISASTERS

TESIS PARA OPTAR AL GRADO DE
MAGÍSTER EN CIENCIAS, MENCIÓN COMPUTACIÓN

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERA CIVIL EN COMPUTACIÓN

CAMILA ANDREA FAÚNDEZ ORELLANA

PROFESOR GUÍA:
JAVIER BUSTOS JIMÉNEZ

MIEMBROS DE LA COMISIÓN:
IVANA BACHMANN ESPINOZA
SERGIO OCHOA DELORENZI
YADRAN ETEROVIC SOLANO

Este trabajo ha sido parcialmente financiado por
CONICYT-PCHA/MagisterNacional/2016-22161328

SANTIAGO DE CHILE
2023

RESUMEN DE LA TESIS PARA OPTAR AL GRADO DE
MAGÍSTER EN CIENCIAS, MENCIÓN COMPUTACIÓN
POR: CAMILA ANDREA FAÚNDEZ ORELLANA
FECHA: 2023
PROF. GUÍA: DR. JAVIER BUSTOS JIMÉNEZ

REDES DEFINIDAS POR SOFTWARE EN LA RED DE INTERNET CHILENA: CONSTRUYENDO RESILIENCIA FRENTE A DESASTRES NATURALES

Chile, un país propenso a desastres naturales como terremotos, tsunamis, erupciones volcánicas y deslizamientos de tierra, a menudo sufre daños generalizados que afectan no solo a las personas, sino también a las redes de Internet.

Los desastres naturales son impredecibles, por lo que una arquitectura de redes que permita la gestión dinámica de la red, como las Redes Definidas por Software (SDN, por sus siglas en inglés), tiene el potencial de ofrecer nuevas soluciones para crear resiliencia y mejorar calidad de servicio (QoS) en escenarios donde las redes de Internet pueden experimentar interrupciones parciales debido a desastres. Sin embargo, la arquitectura SDN fue diseñada originalmente para centros de datos y no para redes de área amplia (WAN).

El objetivo de este trabajo es estudiar la viabilidad de aplicar el paradigma SDN a WAN para diseñar redes resilientes que mejoren la QoS de los usuarios en escenarios de desastre.

La investigación comienza con una Revisión Sistemática de la Literatura que examina la aplicación general de SDN sobre WAN, con un enfoque en las técnicas de QoS y resiliencia, así también como en el impacto específico de los desastres naturales sobre WAN y los beneficios y desafíos que las SDN pueden aportar en tales situaciones. Con base en los resultados de la revisión de literatura, se presenta una discusión que analiza los hallazgos e identifica áreas en las que se necesita más investigación; concluyendo que las SDN se pueden aplicar a WAN y que si bien existen estudios que utilizan SDN para mejorar la resiliencia en escenarios de desastre, existe una falta de investigación en técnicas combinadas y la activación automática de configuraciones de desastre.

Este trabajo propone el diseño y caracteriza un controlador SDN resiliente ante desastres que utiliza tres niveles de priorización basados en IDs, que contiene funcionalidades que pueden no ser fácilmente aplicables en redes tradicionales. Además, se realiza una implementación del mismo utilizando el framework Ryu. Luego, se construye un entorno de pruebas utilizando Mininet y Open VSwitch para simular SDN y probar el controlador implementado en escenarios de redes afectadas por desastres, mediante tests funcionales y no funcionales. Con los resultados de las pruebas, se concluye que el controlador diseñado e implementado entrega herramientas efectivas para construir una red resiliente. El sistema diseñado es extensible y permite la fácil adición de otras aplicaciones y funcionalidades al controlador, lo que lo hace adaptable a nuevas necesidades.

La validación del controlador implementado muestra el valor de las capacidades de monitoreo dinámico y gestión de QoS otorgadas por la arquitectura SDN para su uso en WANs afectadas por desastres.

SOFTWARE DEFINED NETWORKS OVER CHILEAN WIDE AREA NETWORKS: BUILDING RESILIENCE AGAINST NATURAL DISASTERS

Chile, a country prone to natural disasters such as earthquakes, tsunamis, volcano eruptions, and landslides, often experiences widespread damage that not only affects people but also impacts the Internet networks.

Natural disasters are unpredictable, thus a networking architecture that allows dynamic management of the network such as Software Defined Networks (SDNs) has the potential to offer new solutions for improving resilience and quality of service (QoS) in scenarios where Internet networks may experience partial outages due to disasters. However, SDN architecture was originally designed for datacenters and not for wide area networks (WANs).

This research aims to study the feasibility of applying the SDN paradigm to WANs in order to design a resilient network that can enhance QoS for end users during disaster scenarios.

The research starts with a Systematic Literature Review that examines the general application of SDNs in WANs, with a focus on QoS and resilience techniques, as well as the specific impact of natural disasters on WANs and the benefits and challenges that SDNs can bring in such situations. Based on the results of the literature review, a discussion is presented, analyzing the findings and identifying areas where further research is needed. The discussion concludes that SDNs can be applied to WANs and that there are existing studies that use SDNs to enhance resilience against disaster scenarios, but there is a lack of research on combined techniques and automated triggering of disaster configurations.

As a contribution to the field, the author proposes a disaster resilient SDN controller that utilizes three levels of ID-based prioritization. The features of the designed controller are presented in detail and implemented using Ryu, an SDN controller framework, which constitutes the second main contribution of this research.

For the third main contribution, the author builds a testbed using Mininet and Open VSwitch to simulate SDNs and test the implemented disaster resilient controller. The tests conducted include functional and non-functional tests, with a focus on simulating a disaster scenario over the network.

Based on the testing results, it is concluded that the designed and implemented disaster resilient SDN controller effectively provides features and tools to build resilience, as well as implements new techniques that may not be easily applicable in traditional networks. The extensibility of the designed system allows for easy addition of other applications and features to the controller, making it adaptable to evolving needs.

The validation of the designed features of the controller demonstrates the value of the dynamic monitoring and QoS management features provided by SDN architecture in designing a disaster resilient controller for WANs. This research contributes to the body of knowledge on SDNs and their potential applications in disaster resilient networks.

A mis padres, puesto que sin sus esfuerzos e incontables sacrificios este trabajo no hubiera sido posible.

Acknowledgements

I am grateful to my family, especially my mom and Leandro, for their unwavering support and encouragement that helped me persevere through this research. They went above and beyond to create a conducive environment for me to work on my project. Simple acts like cooking me meals were more than enough to keep me going.

I would also like to express my gratitude to my advisor, Javier, for his support, guidance and patience throughout this journey.

Special thanks to my friends for the laughter and good times we shared. Playing board games and ARAMs together provided much-needed relaxation spaces during this process.

Finally, thanks to my pets: Kaiser and Santi; and even the ones that are no longer here: Judy, the parakeets and the finches; and to my plants, for just being there and providing comfort. Additionally, the sacrifice of the plants that didn't make it will be remembered.

Table of Content

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Chile is a country susceptible to natural disasters such as earthquakes, tsunamis, volcano eruptions, landslides, etc. These disasters are known for causing widespread damage in the stricken area, which not only affects people but also impacts the Internet networks.

The Internet telecommunication systems are susceptible to failures under these scenarios. An example is the 2010 Chile earthquake and tsunami that occurred on February 27th, where 64% of Chilean IP addresses were unreachable and 70% of networks were on average 9 hours out of service [48]. Some of the reasons for these failures were the lack of power supply and partial outages.

Natural disasters affect people negatively as it puts their lives at risk. Telecommunication systems help in these situations by allowing the coordination of victims' rescue operations and confirming safety of peers.

1.2 Problem Statement

The traditional networking architecture approach is static and based on hardware network appliances. A networking approach called Software Defined Networking (SDN) proposes a programmable network that allows dynamic network configuration and management through software applications.

Natural disasters are unpredictable, thus a networking architecture that allows dynamic management of the network has the potential to enable the design and creation of new solutions that could help achieve better levels of resilience and quality of service (QoS) in the scenario of a disaster that causes partial outage of the Internet networks. However, the SDN architecture was originally designed for datacenters and not for wide area networks (WANs).

1.3 Purpose statement

The purpose of this work is to study the SDN paradigm and evaluate if it is possible to apply it to WANs with the intention of designing a resilient network capable of improving QoS for the end users in a disaster scenario.

1.4 Research questions

The main research questions that will help to determine the state of the art of SDNs in the context of the problem statement are the following:

1. How can SDNs be applied on WANs?
2. What are the challenges to achieve different levels of QoS or resilience in SDNs?
3. How disasters affect WANs and how SDNs can help mitigate those issues?

And in order to test a developed design, the following research question is needed as well:

4. Which testing frameworks are available to experiment with SDN in WANs?

1.5 Objectives

To achieve the purpose statement in declared in Section 1.3, the specific objectives that need to be fulfilled are the following:

- Perform a Systematic Literature review to determine the state of the art of SDNs in the context of the problem.
- Identify existing frameworks that allow the implementation of SDN solutions.
- Design and implement an resilient SDN solution capable of improving QoS in a network affected by a disaster scenario.
- Evaluate the proposed design by implementing a testing setup.

1.6 Importance of the research topic

Understanding natural disaster scenarios is essential for developing effective emergency response and disaster management strategies. It can help to mitigate the impact of disasters on communities and infrastructure, and ensure that critical services and resources are available when they are needed the most.

This topic has been broadly studied in the context of traditional and current Internet architecture. However, the failures seen in the 2010 Chile earthquake and tsunami show that the Chilean traditional infrastructure was not solid enough to guarantee the operational continuity [48], showing a necessity for disaster resilient solutions.

This work proposes to step out of the box of using the traditional networking approach and not limit the study or development of new disaster resilient solutions to the existing architecture.

The software defined networking architecture is an emerging technology that is becoming increasingly important due to its flexibility to manage networks. One of its key features that allows said flexibility is the ability to maintain a centralized and up-to-date view of the network, which looks like an excellent resource to analyze a disaster situation, assess damage and to respond accordingly in a live manner.

However, the SDN architecture was originally designed for datacenters and not for wide area networks (WANs), thus it is not clear if the current state of this networking paradigm allows it to be applied to WANs in the first place. It is also not clear which tools allow the development of solutions in a SDN architecture. This work aims to provide an answer to these questions and moreover, to develop a disaster resilient SDN solution for WANs and evaluate its performance.

A successful result on applying SDNs for this problem would open an area of research for novel disaster resilient solutions.

1.7 Organization

The paper is organized as follows:

- Chapter 2 "*Systematic Literature Review*" provides a comprehensive review of the existing literature on SDNs, their appliance in WANs and their potential to design disaster resilient networks.
- Chapter 3 "*Review Discussion*" analyzes and discusses the findings from the literature review, identifying research gaps and areas for improvement.
- Chapter 4 "*Framework Design*" describes the key elements of the designed framework and their functions.
- Chapter 5 "*Framework Implementation*" describes the implementation details of the proposed framework, including the tools and technologies used.
- Chapter 6 "*Testing and Results*" presents the experimental setup, the testing methodology and the results obtained.
- Chapter 7 "*Conclusions*" summarizes the findings of the study, highlights contributions, and suggests directions for future research.

Chapter 2

Systematic Literature Review

2.1 Introduction

This Section presents a systematic literature review about SDNs, focusing on research related to its appliance in wide area networks (WANs). This review was performed following the protocol proposed by Kitchenham in [25].

2.2 Review planning

2.2.1 Objectives

The general problem is to evaluate how effective SDNs are in designing resilient networks and improving QoS in the scenario of a WAN affected by a natural disaster.

Therefore, the purpose of this review is to study the state of the art of SDNs, starting with their general application in WANs. It will then focus on QoS and resilience techniques, before delving into the specifics of the impact of a natural disaster on a WAN and the benefits or challenges that SDNs bring.

2.2.2 Research questions

Below are listed the research questions that we want to answer in this review:

1. How can SDNs be applied on WANs?
    (a) What are the challenges in achieving different levels of QoS or resilience?
    (b) How do disasters affect WANs and how can SDNs help mitigate those issues?
2. Which testing frameworks are available to experiment with SDNs in WANs?

2.2.3 Search strategy

The search for primary studies was conducted using the electronic database IEEE. This repository was chosen for its relevance in the field of computer science.

Below is the search pattern applied to the aforementioned database. Logic connectors such as AND, OR, and NOT were used. A bullet below each connector represents elements separated by the respective connector. An asterisk (*) is used as a wildcard that matches any word starting with the specified prefix (e.g., resilien* will match resilience, resilient, etc.).

- AND
  - software defined network*
  - OR
    - quality of service
    - resilien*
  - OR
    - wide area network*
    - disaster*
    - earthquake*
  - NOT
    - survey
  - NOT
    - vehic* network

The aforementioned query represents the idea of searching for "papers that study SDNs, mentions either QoS or resilience, discuss WANs, disasters or earthquakes, are not a survey and do not discuss vehicular networks".

In Table 2.1 the specific query performed on the database is shown.

Repository	**Query**
IEEE Xplore	(("All Metadata":"software defined network*") AND (("All Metadata":"quality of service") OR ("All Metadata":resilien*)) AND (("All Metadata":"wide area network*") OR ("All Metadata":disaster*) OR ("All Metadata":earthquake*)) AND NOT ("All Metadata":survey) AND NOT ("All Metadata":"vehic* network*))

Table 2.1: Query performed on IEEE repository in order to retrieve primary studies for this review.

2.2.4 Study selection criteria

To select papers for study, the following inclusion and exclusion criteria were applied:

- Inclusion criteria
  - The paper is written in english
  - The paper is a primary study (journal or proceeding)
  - The paper was published between 2007 and 2017
  - The paper study SDNs applied in WANs
  - The paper shows conclusions about their study
- Exlusion criteria
  - The paper is not available online
  - The paper focus is vehicular networks
  - The paper is a survey

2.2.5 Study selection process

The papers were selected by the author based on the inclusion and exclusion criteria. The author read both the title and abstract of each paper obtained through the search strategy. Following the study selection criteria, the author decided whether the paper was approved for further steps.

2.2.6 Study quality assessment

In addition to the inclusion and exclusion criteria, it is important to assess the quality of the primary studies. Therefore, the following questions must be answered positively by the study:

- Was the topic of study described appropriately in the paper?
- Were their contributions explained in detail?
- Does the paper present results and conclusions?
- Are the results concrete and comprehensive?

2.2.7 Data extraction

To accurately record the information obtained from the primary studies, a data extraction form was designed. The contents of this form are detailed below:

- Title
- Abstract
- Authors
- Source (repository)
- Publication type
- Research question's answers

The data extraction was performed by the author, who filled in the aforementioned form to record the relevant information obtained from the primary studies.

2.2.8 Data analysis

The research questions were dissected to obtain accurate information. The details of this breakdown are shown in Table 2.2.

Context	Question	Possible answer
General	Main problem studied	Name and description of the problem studied
Applying SDN in WANs	Challenges of applying SDN in WANs	List of challenges
		Potential solutions
	Benefits of applying SDN in WANs	List of benefits
	Challenges to resilience	List of challenges
		Potential solutions
	Challenges to QoS	List of challenges
		Potential solutions
	Which disasters are mentioned?	List of disasters
	Challenges observed once a disaster strikes	List of challenges
		Potential solutions
Testing framework	Real or virtual?	real, virtual, n/a
	Protocols used	List of protocols used
	Tools used	List of tools used
	Data used	real, simulated

Table 2.2: Data gathering form used to extract information from the selected studies.

2.3 Results

2.3.1 Statistical results

The search strategy yielded 57 studies that matched the criteria in the IEEE library. Out of these, 45 studies passed the study selection criteria and quality assessment. A summary of the selected studies per publication year is shown in Table 2.3.

Year	Number of papers published	Number of papers selected
2013	5	3
2014	10	9
2015	13	11
2016	12	9
Total	57	45

Table 2.3: Summary of selected studies per publication year.

Note that even though the study selection criteria includes studies published since 2007, the oldest papers that appear were published in 2013. This aligns with the fact that SDNs were conceptualized around the timeframe of 2008 at Stanford University [36]. Furthermore, initially SDNs were primarily designed for campus networks [37], thus early studies focused on developing the technology. As time progressed, SDN studies shifted towards applying them in various contexts to address diverse problems. In particular, based on the data collected in this study, research on SDN-WANs began in 2013. The number of studies published on this topic reached its peak in 2016, as depicted in Table 2.3.

Table A.1 shows a summary of the 45 articles that were selected for the data extraction process and their main research topic.

2.3.2 Data analysis results

The studies that passed the selection criteria and quality assessment share a common topic: SDNs applied to WANs. However, the data analysis breakdown reveals that this field of study encompasses a wide range of sub-problems. In this section, we will first present the general concept of SDN and then discuss the main problems studied by the papers included in this review.

2.3.2.1 SDN overview

The main idea behind the development of SDN is to separate the data plane from the control plane, thereby creating an abstraction between the controller (control plane) and the network elements [44].

Decoupling the control and forward (data) planes makes the network programmable and provides logic centralized control [59] through the controller. Therefore, the SDN controller is often referred to as the brain of the network [56], because it has a global and accurate view of the network and orchestrates all operations in the data plane [10].

The centralized controller(s) can be programmed to manage network resources and oversee operations taking place at all network devices under its supervision [24].

To manage the data plane, communication between the control plane and the data plane is facilitated through a southbound API such as OpenFlow [27].

OpenFlow is a protocol that allows a SDN controller to communicate with an OpenFlow Logical Switch [18]. It provides a method to establish a controller-switch connection (channel) which is used to send messages between these entities. Depending on the situation, these messages can be initiated by either the controller or the switch.

An OpenFlow Logical Switch contains of one or more flow tables (which perform packet lookups and forwarding), a group table, a meter table, and one or more OpenFlow channels which connect it to an external controller. Figure 2.1 illustrates the aforementioned switch components.

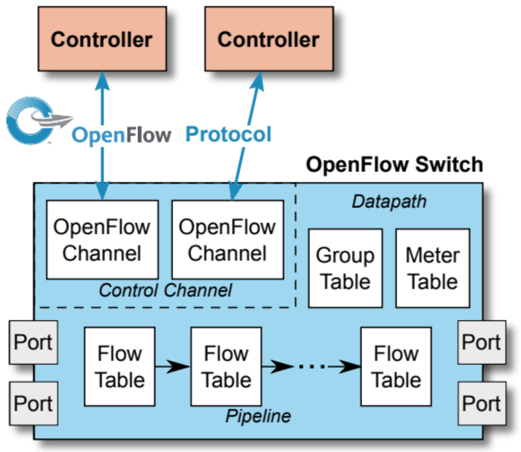


Figure 2.1: Main components of an OpenFlow Switch [19].

A traditional switch uses a routing table to determine the appropriate forwarding path for incoming packets. In contrast, an OpenFlow Switch processes all incoming packets through the OpenFlow pipeline. The OpenFlow pipeline consists of one or more flow tables, each containing flow entries.

The pipeline processing always begins with ingress processing at the first flow table. When a packet is processed by a flow table, it is matched against the flow entries in that table to select a flow entry:

- If a matching flow entry is found, the instruction set included in that flow entry is executed. These instructions may include directing the packet to another flow table, where the same process of matching and instruction execution is repeated. Pipeline processing can only proceed forward and does not allow backward traversal.
- If the matching flow entry does not direct packets to another flow table, the current stage of pipeline processing terminates at this table.Tthe packet is then processed with its associated action set, which typically involves forwarding the packet to its intended destination.
- If a packet does not match a flow entry in a flow table, it defaults to the table miss flow. The instructions defined in the table-miss flow entry in the flow table determine how to process unmatched packets. Useful options include dropping them, forwarding them to another table or sending them to the controller via packet-in messages over the control channel.

A packet-in message is used to notify the controller about a new flow, and based on its configuration, the controller can make decisions such as installing new flow entries, tables, etc.

The pipeline within the switch is defined by the controller immediately after establishing the controller-switch connection. Consequently, it is the controller who configures all the rules installed on the switch. This approach transforms the switches in the network into simple packet forwarding devices [29].

Lastly, this architecture is complemented by the application plane. The application plane manages the control plane through applications, also referred to as controller applications [20]. These applications request information from and provide instructions to the control plane. The application plane interacts with the control plane through a northbound API. These applications serve as the programmable components of SDNs, enabling dynamic network management.

The three planes of the SDN architecture: data plane, control plane and application plane, are summarized in Figure 2.2

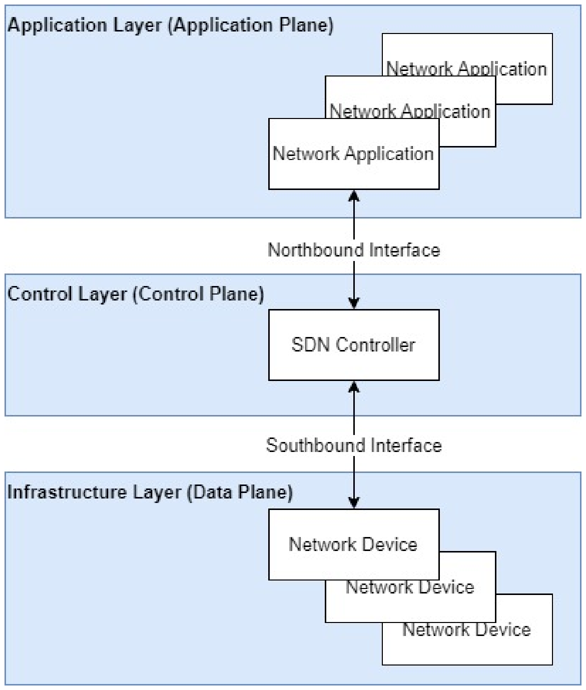


Figure 2.2: Main functional layers or planes of the SDN architecture.

The SDN architecture makes the networks programmable and provides centralized control of multi vendor environments [29].

2.3.2.2 SDN applicability on WANs: Challenges overview

The primary motivation for proposing the SDN architecture in WANs is to leverage the advantages of network programmability and management offered by this innovative architecture. As previously mentioned in Section 2.3.1, SDNs were initially designed for campus networks, i.e. local area networks (LANs). LANs and WANs differ in terms of their intended use and scale. This Section will highlight the challenges that SDNs must address to be effectively implemented in WANs.

The first challenge identified is *scalability.* As mentioned in [37], scalability becomes a concern when applying SDNs to WANs due to the centralized nature of the original SDN/OpenFlow model. The issues highlighted are if the controller has the capacity of handling a huge number of OpenFlow requests, and if it is possible to keep a global view of a WAN inside of it. The concerns raised include whether the controller can effectively handle a large volume of OpenFlow requests and whether it can maintain a global view of the entire WAN within its control. Similarly, [52] also point out that early deployments of SDN relied on physically-centralized control-plane architectures with a single controller, which can lead to scalability, *performance*, and *reliability* problems in a large network deployment such as WANs. They also emphasize that the centralized design is vulnerable to disruptions and attacks, particularly due to having a *single point of failure.* Both studies aforementioned provide the same solution, which is to rely on multiple controllers to manage the network. This approach is known as a distributed (or decentralized) control plane, and is further analyzed in Section 2.3.2.3.

Following the distributed control plane approach, another challenges that arises are determining the *optimal placement of multiple controllers*, the *required number of controllers*, and *how to connect them to the switches* to achieve satisfactory levels of reliability and performance [34]. This particular area of study is referred to as the Controller Placement problem and is discussed in detail in Section 2.3.2.4.

Another important aspect to consider is that the Internet infrastructure face numerous *challenges in normal operation*, such as power outages, link failures, device failures, and more [37]. Some of these challenges are not experienced in controlled environments such as campus networks or data centers, which were the original deployment scenarios for SDNs. These challenges in WANs translate into multiple issues for SD-WANs. Firstly, the physical separation between the control plane and the forwarding plane reduces the reliability of their communication. If a failure occurs that disconnects their communication channel, the forwarding plane could be disabled, leading to packet loss, degraded performance, and reduced resilience. Secondly, the controller in an SD-WAN environment deals with a constantly changing network. This includes component failures and configuration changes, which are particularly common and extensive in disaster scenarios. Maintaining an accurate and up-to-date view of the network becomes crucial so that the controller can make informed management decisions in a timely manner. These challenges highlight the need for resilient and adaptive mechanisms in SD-WANs to ensure reliable and efficient network operation, particularly in the face of disruptions and dynamic network conditions. These challenges highlight the *need for recovery strategies* in the event of network changes, as well as in critical scenarios such as disasters. These strategies aim to enhance network resilience and ensure quality of service (QoS) for end users. In Section 2.3.2.5, we will delve deeper into the review of these strategies, which focus on improving network resilience and QoS in SD-WANs.

Securing communications in WANs is another challenge that was not initially addressed by the original concept of SDNs designed for LANs. WANs, with their widespread design and connection through the public internet, introduce additional security considerations. Potential solutions to this challenge are reviewed in Section 2.3.2.7.

The final challenge of deploying SDNs in WANs is the presence of *multiple Internet Service Providers (ISPs)* and the variability in their domain configurations across the WAN. This lack of uniformity makes coordination between ISPs non-trivial and introduces complexities in managing the network [37]. Additionally, there is a possibility of interconnecting SDN domains with non-SDN domains [8], further adding to the challenges of network management and coordination. These limitations and their implications are analyzed in Section 2.3.2.10.

2.3.2.3 Distributed Control Plane in WANs

SDNs advocate for a logically centralized control plane, which can be physically composed of either a centralized controller or distributed physical controllers. Distributing the control plane is often necessary to achieve scalability in large networks, enhance resilience, and prevent a single controller from becoming a point of failure [41].

Phemius et al. [46][47] propose DISCO, a Distributed SDN Control plane for WAN and overlay networks. In their proposed architecture, each DISCO controller is in charge of a SDN domain. A DISCO controller contains intra-domain and inter-domain functionalities. The intra-domain module monitors the networks and manages flows (i.e. data plane management). The inter-domain module communicates with neighbor domain controllers to exchange aggregated network-wide information through two key components: (1) a messenger module which provides a control channel between neighboring controllers and (2) agents that use this channel to exchange network wide information with other controllers. The details of the intra-domain work was extended from a previous study [45] (not included in this review), which shows that data plane management does not care if the control plane is composed by single controller or by multiple distributed controllers.

By having a number of high performance controllers and successful deployment, the scalability problems are solved and the focus is directed at improving performance and resilience of SDN WANs [37].

Controller deployment in terms of number and placement inside the network is further reviewed in the next Section. Performance and resilience in SDN WANs is reviewed on 2.3.2.5.

2.3.2.4 Controller placement problem in WANs

The controller placement problem (CPP) indicates the number of required controllers to handle the switches' demands as well as their location (in the network topology) in an efficient and cost effective manner [3]. This problem reduces to a Facility Location Problem and is proven to be NP-Hard [44].

The general controller placement problem is well explored and now more and more authors have addressed the Facility Location problem in the context of controller placement in SDN networks [27]. Furthermore, this review gathered five studies ([34], [52], [52], [54], [44]) which analyzes the problem specifically in the context of SDN-WANs, which accounts for 11.1% of the papers studied in this review.

The reason this problem continues to be studied is because depending on the constraints applied to formulate the problem, different algorithms are created to account for them.

In [27] the authors focus on two constraints: (1) minimizing the average latency between the nodes and their controllers and (2) minimizing the load imbalance among controllers; to propose an algorithm to solve the CPP.

In [44], authors propose a formulation of the problem that aims to minimize the number of controllers taking into account the maximal allowed latency between routers and controllers, the maximum allowed latency between controllers, and the load balancing constraint between the controllers sub-domains.

In [54], the authors formulate the problem from a resilient point of view by taking into account the capacity of the controllers as well as the demands of the switches, and also assuming in-band control (i.e. no dedicated links between controllers and switches).

In [34], the authors provide an algorithm to solve the CPP in large scale SDNs under the following constraints: controllers placed on existing nodes location, controller-switch channel using existing transport network, traffic flow considered as existing demand on the network, priority on nodes with highest bandwidth processing and latency calculated as propagation latency.

In [52], the authors tackle the problem from a disaster resilient point of view. They design the control plane considering survivability requirements of the network, i.e. they decide on the total number of controllers and their placement in the topology based on the following constraints: number of controllers guaranteed per switch, number of controllers active at any time, maximum number of switches assigned per controller, P_{ij} set of paths available between nodes i and j of the network, U_p^y probability of path p surviving disaster y and a set of disasters.

2.3.2.5 Resilient SD-WANs

With a decentralized approach, the failure of a controller no longer constitutes a single point of failure as long as mechanisms are in place to ensure continuity of service.

The OpenFlow protocol allows a switch to establish communication with multiple controllers since version 1.2. Having multiple controllers improves reliability, as the switch can continue to operate in OpenFlow mode if one controller or controller connection fails. The hand-over between controllers is initiated by the controllers themselves, which enables fast recovery from failure and controller load balancing [19].

In the context of fast recovery, Obadia et al. [41], present two failover mechanisms to migrate control of orphan switches to other active controllers. The strategy involves the active controllers discovering the failure of its neighbor controllers.

In the context of a controller channel failure, Nguyen et al. [36] propose using multiple paths for each controller-switch communication and argue that having a single path for OpenFlow channel decreases reliability of the SDN WAN. The authors propose using multipath in-band control with TCP/IP connectivity due to cost-effectiveness instead of building and maintaining a dedicated out-of-band control channel. They evaluate their work on [42].

In the context of load balancing, Singh et al. [53] propose an algorithm to optimize load balancing among controllers by taking the decision of when to migrate a switch from one controller to another based on the controllers response time, in order to minimize it.

Chang et al. [10] design the SDN control plane with the purpose of addressing the high traffic load on edge routers in WANs. Their approach focuses on dynamically controlling the traffic at ingress devices according to the overall state of the network. They also propose a method to logically enlarge the network buffers of routers through global packet caching.

2.3.2.6 Routing algorithms in SD-WANs

The routing algorithms found in this review for SD-WANs are mostly traditional algorithms for WANs adapted to the SDN architecture. One of the main features of SDN that benefits the implementation of these algorithms is having a centralized and global view of the network, which reduces the communication overhead needed to achieve the same view in traditional distributed approaches [3]. In this review, we found two articles that completely focus on routing algorithms in SDNs.

Firstly in [57], the authors use a modified Dijkstra algorithm to route new flows and ensure maximal resource utilization. They implement this algorithm as a controller application.

In [3], the authors proposed a routing algorithm called Shortest-Feasible OpenFlow Path (SFOP) to find the optimal path between nodes, reduce latency, and enhance resource utilization without increasing computation complexity. They use a modified shortest-widest path algorithm in conjunction with the inputs: (1) residual bandwidth and (2) feasible bandwidth. These inputs are obtained from statistics collected from the OpenFlow interface.

2.3.2.7 Security in SD-WANs

This literature review did not recollect articles that specifically address SDN architectural security in WANs. However, two studies address how to secure user communications through SD-WANs.

In [56], the authors describe a method for establishing a secure overlay network over SD WAN, with the purpose of having a safe mechanism for data transmission between two locations and prevent interception by unauthorized third parties. They propose a SDN control plane, encrypted links through VXLAN tunneling and virtual switching using Open vSwitch.

Authors in [15] propose an SDN framework for airborne connectivity. The key features include securing cockpit traffic through tunneling and separating it from cabin traffic, as well as prioritizing cockpit traffic over cabin traffic.

2.3.2.8 Disaster scenario in traditional WANs

In order to design recovery strategies for disaster scenarios, it is essential to characterize these scenarios to understand how they affect WANs. This review identified eight different studies that provide detailed information about specific events, their consequences, and the faults they generate in WANs.

Concrete events analyzed by the authors of studies gathered by this review are detailed below:

- World Trade Center disaster in 2001 [29].
- Tsunami in Asia-Pacific (2004) [29].
- Hurricane Katrina in the United States of America in 2005 [29].
- Sichuan earthquake in 2008, where around 30,000 km of fiber optic cables and 4,000 telecom offices were damaged [52].
- The Great East Japan Earthquake and Tsunami in 2011 [42][51][50], where around 1,500 telecom buildings faced long power outages by the main shock on March 11th. There was a widespread blackout over northern and central Japan [50]. While most were fixed, 700 telecom buildings experienced power outages by the aftershock on April 7th, 2011 [52]. In addition, many Japanese coastal areas were geologically isolated [50].

Furthermore, other events mentioned in [58] and [38] include earthquakes, hurricanes, floods, fires, power outages, electronic attacks and intentional attacks.

In general, natural disasters are characterized as unexpected [38], unpredictable, and disruptive [58]. During major disasters, it is common to observe significant network congestion and experience service outages due to infrastructure damage [26]. [26] also highlights that emergency scenarios can lead to high levels of packet loss, which can be attributed to network congestion or infrastructure damage.

In the context of Internet services, infrastructural damage caused by a disaster results in widespread node and link failures [38], failure of wired communication network, power supply failures in communication equipment, equipment failures, and high levels of congestion in the communication network [51].

One characteristic of natural disasters is their potential to simultaneously destroy multiple network facilities within a specific geographical area [58]. Such disasters may cause partial or complete damage to the terrestrial Internet infrastructure, leading to an inability of the network to handle the usual surge in resource demand and provide the required bandwidth and quality of service (QoS) guarantees [26]. Under these conditions, emergency communications reporting catastrophic or emergency events may experience packet loss, resulting in incomplete and/or delayed communications [29].

The loss of transmission capability of disaster information can cause delays in rescuing victims, directing people to the shelters, confirming safe evacuations and providing urgent medical treatment [50]. A timely and fast response is crucial in disaster situations as a significant portion of fatalities occur within the first few hours of the disaster [29].

On top of high congestion and large scale node and link failures, Ogawa et al. [42] identify the following limiting factors:

- Some network administrators change the network configuration manually. If the administrators of information infrastructures are unable to reach the place of the information infrastructures at disasters, network configurations can not be changed. This highlights the need for automated network operations.
- Network configuration technologies should not control the networks of other organizations. Therefore, changes in configuration can only be performed in the controlled domain.
- Latest research shows the existence of disaster areas where the Internet is available even immediately after the earthquake. Therefore, connections among the different available parts of communication networks can provide some level of communication in damaged areas at disasters, as long as these surviving areas can be reconnected.

2.3.2.9 SD-WANs for disaster scenarios

The first challenge for SD-WANs is whether the SDN architecture design is able to endure disasters. Thus, K. Nguyen et al. [37] investigate the deployment of SDN in a realistic network topology and simulate a disaster scenario. They evaluate two important issues: the communication between networking devices and controllers, and the recovery process after link failures. The evaluation results confirm the applicability of SDN on disaster-resilient WANs.

The second challenge is to detect the disaster scenario, which is necessary to enable restoration techniques. While manual user input can be used to notify the controller about a disaster, there are alternatives, such as using sensors that provide information to the controllers or leveraging SDN network monitoring features ([50], [39]), to trigger disaster configuration. OpenFlow provides built-in messages that allow the switch to communicate link failures to the controller. It also offers messages to verify the liveness of a controller-switch connection and measure latency or bandwidth [18]. These combined features enable the development of disaster detection techniques.

Benet et al. [6] consider techniques that utilize OpenFlow to detect link failures as reactive link failure recovery. They argue that although accurate, the process of (1) the switch notifying the SDN controller and (2) the controller reacting and installing new forwarding rules might take over 150ms. Therefore, the reaction time, from when the disaster occurs until it is detected by the controller, must be taken into consideration.

Then, the next challenge is to address changes in the network caused by multiple component failures and to design recovery strategies to handle the disaster scenario.

In [5], it is argued that reroute techniques based on shortest path restoration do not guarantee the fastest recovery time in scenarios involving multiple node failures. As a result, they formulate the problem of flow restoration with minimum operation cost in OpenFlow networks. They considered the number of flow operations required for path restoration and developed rerouting algorithms. Additionally, they prove that the problem of discovering the path with the minimum operation cost has a polynomial complexity.

In [58], the authors design a control plane with the objective of achieving fast recovery and improved reachability. They employ a combination of pre-computed backup topologies to rebuild failed connections and propose a controller module to identify new paths when necessary. The pre-computed backup topologies help reduce the routing computation load on the controller. Their approach is implemented by utilizing the multiple tables pipeline processing and fast failover group tables of OpenFlow.

V. Nguyen et al. [39] propose a method for selecting appropriate backup paths to reroute flows based on location trustworthiness. Their Location Trustworthiness framework gathers disaster data from sensors and calculates trustworthiness values for locations, links, and switches. They design their framework within an SDN architecture using OpenFlow switches and develop a controller application to implement the framework.

Finally, the last challenge to address is high congestion. In this scenario, authors propose methods to ensure the QoS for critical communications.

One technique that is applied is prioritizing important traffic. The authors in [60] emphasize that prioritizing crucial traffic, such as mail, phone, or social networking services (SNS), instead of mobile video traffic, has been proven to be effective in disaster scenarios. In [23] it is explicitly stated that resource prioritization may be required in order to maintain a level of QoS in an emergency situation.

Manic et al. [29] propose a framework for Emergency Communication Systems (ECS) using SDN. In their design, each service or application is allocated a dedicated "channel" utilizing VLANs for isolation. They employ SDN and OpenFlow to achieve dynamic prioritization within the framework.

Ogawa et al. [42] propose a method to prioritize crucial communications that are associated with disaster areas. Their approach utilizes OpenFlow functionalities to assign "Disaster IDs" to packets related to disaster areas, which allows the switches to identify and prioritize them.

Their method relies on having an IP address and geographical location mapping database at the controllers. When a switch receives a new flow, it sends the first packet to the controller. Then, the controller determines if this flow is related to a disaster area by examining the source and destination IP addresses. Subsequently, the controller installs a rule in the switch to assign a disaster ID (or a "non-disaster" ID) to packets belonging to that flow. This enables all network equipment to make processing decisions based on the IDs without constantly consulting the controllers.

The switches handle and prioritize packets based on IDs through pre-installed set rules by the controller. Thus, reading and prioritizing IDs from existing flows does not require querying the controller or the database.

2.3.2.10 SD-WAN and multiple Internet Service Providers

One of the main advantages of SDNs is that this network model is dynamic and can be easily controlled by the administrator solely through the control plane [53]. However, in WANs there are multiple ISPs, and each ISP implements its own set of policies. The studies collected in this review make different assumptions regarding domains and their configurations.

Most studies in this review don't address the multiple service provider scenario and authors test their proposals inside a single domain configured by them.

Some authors assume that SDNs are deployed throughout the entire WAN. For instance, in [31], it is assumed that the entire transport network is managed as a unified logical forwarding domain, where the SDN controllers make the forwarding decisions. Similarly, in [33], authors assume that each WAN service provider operates its own SDN platform, but the SDN controllers are interconnected, exchanging information about their respective domains. The proposal by Ogawa et al. [42], which is reviewed in Section 2.3.2.9, also requires organizations to adopt the same SDN configuration in order to facilitate cooperation.

The aforementioned setup allows for collaboration between domains. However, even if SDNs are implemented in all WAN domains, challenges can arise due to potentially having different configurations. In [24], the authors address the issue of using different OpenFlow versions among domains, particularly in the context of quality of service (QoS). The OpenFlow protocol has supported per-flow QoS since version 1.3, whereas older versions do not have the capability to manage per-flow bandwidth. To overcome this, their proposal functions as a proxy between SDN controllers and network devices, monitoring and intercepting OpenFlow messages to use Open vSwitch Database Management protocol to manage bandwidth at the switch's output ports.

The scenario where no assumptions can be made about neighboring domains is only addressed in one study. Specifically, the authors in [8] acknowledge the possibility that SDN domains may be interconnected through non-SDN domains. In response to this scenario, they propose a northbound interface to allow SDN domains management through Network Function Virtualization (NFV) orchestrator to account for this scenario.

2.3.2.11 SD-WAN implementation and testing

The studies included in this review use a variety of tools and protocols in order to implement and test their proposals. This Section summarizes the tools and protocols used specifically for the context of implementing and testing an SD-WAN. Tools which were used for other purposes (e.g. to test algorithms runtime) were omitted.

2.3.2.11.1 SDN Controller Platforms

Six different controllers were used by authors in this review to perform their experiments:

- NOX is the first SDN controller developed and is only used by one study in this review [58]. It's open source and written in C++.
- POX is a sister project of NOX written in Python, and it is used by four papers in this research ([24], [3], [30], [37]). It's also an open source project.
- ONOS controller is an open source project written in Java and is used in two studies ([16], [8]).
- Ryu controller is a component-based SDN framework. It is written in Python and open source. One study uses it to implement their controller [9].
- OpenDaylight is an open source Java-based controller used by five studies in this review ([59], [39], [49], [21], [6]).
- Floodlight is a community developed, open source, Java controller. It is used by ten studies [53, 43, 38, 33, 25, 22, 19, 11, 9, 2], making it the most used controller in this review.

The reasonings of why a controller was chosen over another are only listed in three studies:

- Authors in [57] mentioned that they initially used NOX for their prototypes. However, since NOX development has been stopped on GitHub, they migrated to Floodlight because it has an active community of contributors.
- In [24], authors include both POX and Floodlight in their experiments due to being "the most popular SDN controllers".
- In [26], authors decide to use Floodlight because of it being "relatively easy to use".

And after deciding on a controller, two authors mentioned difficulties in working with it:

- Authors in [56] mention that Floodlight does not have good documentation for configurations in the v1.1 release, while also highlighting that high availability features are missing in the same version.
- In [14], authors also highlight that Floodlight GUI is not reliable and does not update correctly.

In summary, it is shown that all authors in this review decided to use open source controllers. The only factors mentioned that led to choosing one controller over the other are popularity, ease of use and having an active development community.

2.3.2.11.2 Southbound API

As previously mentioned in Section 2.3.2.1, a southbound API in the context of SDN is used to communicate controllers and switches or routers of the network. In this review, sixteen papers specified the southbound API chosen, and all sixteen of them used OpenFlow. In particular, ten articles did not specify the exact version ([51], [56], [14], [21], [29], [8], [8], [30], [41], [47], [46]), two specified they used OpenFlow 1.0 ([24], [37]) and four studies used OpenFlow 1.3 ([60], [49], [26], [58])

2.3.2.11.3 Real and virtual testbeds

Regarding testbed used or constructed, most authors chose to simulate virtual networks. In particular, twenty one studies chose to simulate virtual networks [4], [29], [44], [24], [39], [33], [14], [49], [53], [26], [3], [8], [58], [37], [41], [47], [46], [10]. [6], [28], [7], while only seven [50], [35], [14], [51], [16], [52], [7] use real testbeds. Authors in [7] use a combination of both real and simulated testbed for their experiments.

2.3.2.11.4 Topologies

In terms of topologies used, the following authors only mentioned the amount of nodes in their topology:

- The authors of [39] mentioned a topology with 40 nodes and 70 links.
- In [49], the authors utilized a topology with 4 nodes, each representing a different site.
- A physical network with 14 nodes and 32-Gbps link capacity was used by the authors in [52],
- In [14], the authors employed a real network topology that included 5 WAN switches, 3 ships, and 2 data centers at shore.
- A WAN topology with 10 routers and 2 controllers was used in [53].

Authors in [37], [27] and [54] used topologies obtained from the Internet Topology zoo, which are based on real topologies.

Other authors used known topology types:

- Authors in [41] used a linear topology of N switches and 3 controllers.
- In [6], two k=4 fat tree topologies were used.

Other studies such as [58] and [7] use random topologies, while authors in [47] and [10] both emulate enterprise production networks.

2.3.2.11.5 Network emulators and simulators

Different tools exist which allow the creation of virtual networks. This Section summarizes the ones used by the authors in this review.

- Mininet is a network emulator which is able to create a network of virtual hosts, switches, controllers, and links [13]. It is the most used network emulator in this review with fourteen studies using it ([24], [39], [33], [14], [49], [53], [26], [3], [8], [58], [37], [41], [47], [46]).
- ns-3 is a discrete-event network simulator [40] and is used by one study in this review [10].
- CORE emulator is a tool for building virtual networks [12] and is used by one study [6].
- netem provides network emulation functionality by emulating properties of WANs [17]. It is also used by one study [41].

A relevant note is that authors in [41] combined both Mininet and netem in their simulations.

2.3.2.11.6 Virtual Switch

In the case of simulated or emulated SDN testbeds, these need virtualized switches that are compatible with the SDN architecture. The list below summarizes all the virtual switches found that were used in the studies gathered by this review:

- FLARE switch, which was used by one study [60].
- OpenVSwitch, which was used in ten studies ([42], [51], [56], [20], [14], [26], [29], [41], [47][46]).

2.3.2.11.7 Other tools

Some additional tools that were utilized by multiple authors include:

- iPerf, used in [60], [39], [20], [53] and [21].
- matlab, used in [31], [27] and [54].

Chapter 3

Review Discussion

3.1 Introduction

This Chapter proposes a discussion based on the results of the Systematic Literature Review performed on Chapter 2. The discussion will focus on analyzing the results of the review, formulate overall answers to the research questions and identify areas that lack existing studies; with the purpose of generating novel contributions and develop the core ideas that a resilient SDN framework designed for Chilean networks should consider and implement in order to be effective.

3.2 Applicability of SDN on WANs

Based on the conducted review, it is concluded that SDNs can be applied over WANs and is proven to be an effective technology. The studied articles show the original SDN architecture was studied and enhanced in order to provide a solution and support the new challenges present on WANs.

3.3 Security on WANs

As it was previously reviewed in Section 2.3.2.7, security in the context of protecting the SD-WAN architecture is not covered by this review. This is not a surprise to the author considering the focus of the systematic literature review was resilience and QoS over SD-WANs, and securing SD-WANs is out of the scope of this work.

3.4 Disaster resilient WANs

This literature review compiled a set of techniques which, by taking advantage of the SDN architecture, are able to provide resilience and QoS on WANs affected by disaster scenarios.

Authors of the reviewed studies propose methods in isolation, which is not surprising given the publication format of the articles. Combining techniques to ensure resilient SD-WANs is not explored in the studies gathered by this review, which becomes an open field for contributions. Thus, in this work we will explore combining techniques, and the reasons of which methods were chosen are going to be discussed in the following Sections.

3.4.1 Disaster detection in WANs

This literature review confirmed SDN monitoring capabilities provide useful tools to design disaster detection techniques. However, other than mentioning sensor usage or monitoring link and switch failure as a strategy, no further implementation details are provided.

Questions such as under which conditions should disaster configuration be triggered and how a controller can be automatically re-configured to switch from normal execution into disaster configuration are not answered. Moreover, when should normal execution be restored is another topic not covered by studies in this review.

Disaster detection is key in order to develop a SD-WAN disaster resilient framework. Thus, this work will propose an answer to these questions by developing a solution using the SD-WAN architecture and providing details and considerations about how to implement it.

3.4.2 Prioritization

Prioritization was agreed on and proven to be effective as a QoS measure to ensure critical communications in highly congested networks, which are characteristic of disaster scenarios.

3.4.2.1 ID usage to determine prioritization

ECS prioritization was proposed by Manic et al. [29]. However, their prioritization method of providing dedicated channels to each prioritized service is not scalable and can only ensure channels to a limited amount of services.

Prioritization using IDs, proposed by Ogawa et al. [42] is an idea that can be enhanced. In their proposal, prioritization IDs are assigned to communications related to disaster areas. In this work we propose to enhance ID attachment by also including ECS messages into the list of important messages.

It is also worth noting that ECS messages are only emitted by a select amount of senders and it can be argued that are more important than a randomly picked message related to a disaster area. Due to this reason, we decided to assign a different disaster ID to ECS messages, which allows to (1) differentiate them from other disaster related messages and (2) to manage them differently and provide a higher level of prioritization.

3.4.2.2 Prioritization based on IP address versus packet content

Ogawa et al. [42] defines “messages related to disaster areas” as packets that have their source IP address or destination IP address located in a geographical area affected by a disaster. They also review the possibility of attaching disaster IDs based on packet contents. Ogawa et al. analyzed the latter method and they encountered a flaw which corresponds to false rumors. False rumors related to a disaster would also get prioritized with a packet content analysis approach. They end up not implementing ID attachment based on packet content and label it as a potential future work.

We decided against using packet content as a method to determine disaster IDs attachment based on Ogawa’s review and also based on the considerations of controller reaction time highlighted by Benet et al [6]. Requesting the controller to decide if an ID should be attached to a flow already has a baseline reaction time of one round time trip (RTT) in the controller-switch channel, thus the time the controller needs in order to process the request must be minimized. Packet content analysis, depending on how accurate it needs to be, requires more processing power than querying an IP database, which is why prioritization based on IP addresses is considered as the better alternative by the thesis author.

3.4.2.3 Challenges of prioritization based on IP address and geographical location

Prioritization based on IP address is not a perfect solution and it is important to understand the challenges and difficulties that come by using this method.

A flaw in prioritization of messages based on geographic location affected by a disaster is that not all communications coming from or going to disaster areas are necessarily disaster related, thus ending up prioritizing messages that are not relevant. Ogawa et al. do not mention nor discuss this potential issue.

Another challenge, which is mentioned by Ogawa et al. [42], is that communications that go through servers such as e-mails and social media do not get prioritized by their method. They specifically mention “the packets that the recipients send and receive to and from the e-mail servers do not have source or destination IP address that is related with disaster areas. Therefore the packets are not prioritized.”.

We argue that while the statement is correct, it is not an issue and on the contrary, is a desirable behavior. To explain this reasoning, consider the following scenario: an user located in a geographical area affected by a disaster scenario sends an email to a family member. In this scenario, the participants are:

- *Sender*, which will send an e-mail directed to the recipient. The e-mail will pass through an e-mail server before arriving at the recipient.
- *E-mail server*, which will receive the e-mail from the sender and will pass it down to the recipient upon request.
- *Recipient*, which will request potential e-mails to the e-mail server and will eventually receive the e-mail sent by the sender.

This scenario experience variations in terms of which messages get prioritized depending on the location of the participants:

1. All participants are located in areas affected by a disaster. This is the trivial case where all communications get prioritized, as shown in Figure 3.1. This is because all source and destination IP addresses are located in the area of the disaster and all packets of the communication will get a disaster ID assigned, which means all packets get prioritized.
2. The sender and the e-mail server are located in areas affected by a disaster, while the recipient is not. This case is shown in Figure 3.2. Communication from the sender to the e-mail server is prioritized because all packets get assigned their disaster ID. The main difference from case one is that the communication between the e-mail server and the recipient is partially prioritized, which means the only packets that get prioritized are the ones that traverse links inside the area of the disaster. This works as follows: when the recipient sends a request to the e-mail server to retrieve new e-mails, this request will first travel through switches and links that operate normally, as this area of the network is not affected by the disaster. Eventually, the request will reach a switch that is operating in special mode due to being located in an area affected by a disaster. This switch will attach a disaster ID to the packets of the request due to the destination address (IP address of the e-mail server) being in a disaster area. This means that the request will get prioritized while it travels to the e-mail server. Upon receiving the request, the e-mail server will answer by delivering the e-mail sent by the sender to the recipient. As the source IP address (the e-mail server) is in a disaster zone, disaster IDs will also be attached to this new flow. This means the e-mail will get prioritized through disaster affected links until it exits the critical area. Now under normal operation mode, the e-mail will be sent as usual (as any other communication) to the recipient. This makes sense because if the recipient is not in (or near) an area affected by the disaster, communications around them should not be negatively affected by it. Thus, prioritization in the network around the recipient is not needed because it is operating under normal circumstances and packets should arrive as expected.
3. Only the sender is located in an area affected by a disaster and both the e-mail server and the recipient are not. This case is shown in Figure 3.3. In this case, communication between the sender and the e-mail server gets partially prioritized, and communication between the e-mail server and the recipient is not. Due to the same reasons explained in case two, partial prioritization between the sender and the e-mail server is expected: packets get prioritized only in the links that require it. The network between the e-mail server and the recipient are not affected by the disaster, thus it is correct to not prioritize the packets as they are under a normal operating network.
4. Sender and recipient are in disaster areas, while the e-mail server is not. This case is shown in Figure 3.4. In this case all communications are partially prioritized, with both ends (sender to e-mail server, and e-mail server with recipient) explained in aforementioned cases two and three.

The scenario where the recipient is the user affected by the disaster can be extrapolated from the cases explained above by switching the roles of the sender and recipient. The final scenario where only the e-mail server is located in a disaster area also generates the case where all packets are partially prioritized, which is similar to case four.

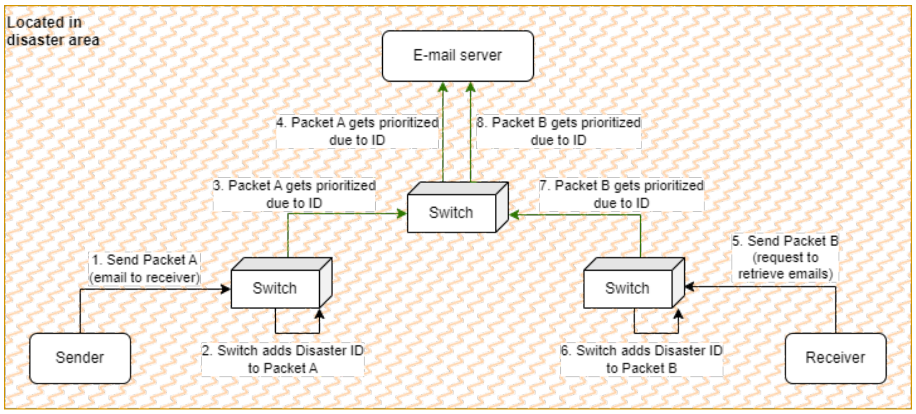

Located in disaster area
E-mail server
4. Packet A gets prioritized due to ID
8. Packet B gets prioritized due to ID
3. Packet A gets prioritized due to ID
Switch
7. Packet B gets prioritized due to ID
1. Send Packet A (email to receiver)
Switch
Switch
5. Send Packet B (request to retrieve emails)
Sender
2. Switch adds Disaster ID to Packet A
6. Switch adds Disaster ID to Packet B
Receiver

Figure 3.1: Example of packets being prioritized depending on the location of the participants. As all participants are located in a disaster area, all communications get prioritized.

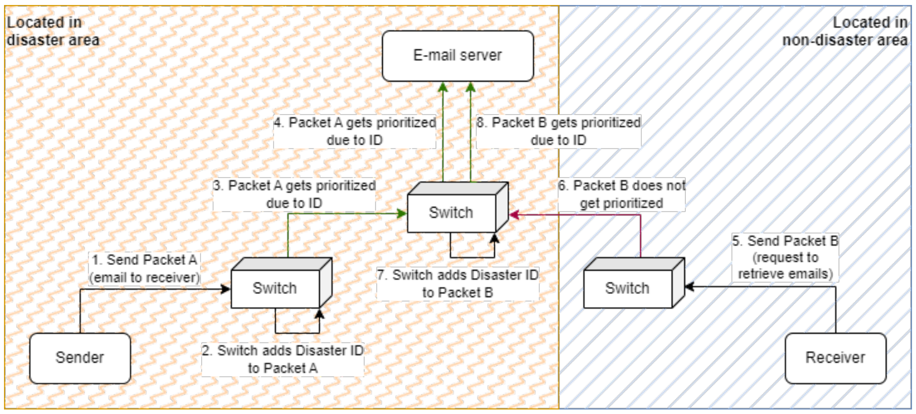

Located in disaster area
Located in non-disaster area
E-mail server
4. Packet A gets prioritized due to ID
8. Packet B gets prioritized due to ID
3. Packet A gets prioritized due to ID
Switch
6. Packet B does not get prioritized
1. Send Packet A (email to receiver)
Switch
7. Switch adds Disaster ID to Packet B
Switch
5. Send Packet B (request to retrieve emails)
Sender
2. Switch adds Disaster ID to Packet A
Receiver

Figure 3.2: Example of packets being prioritized depending on the location of the participants. Communications between sender and e-mail server are fully prioritized, while the communication between recipient and e-mail server gets partially prioritized.

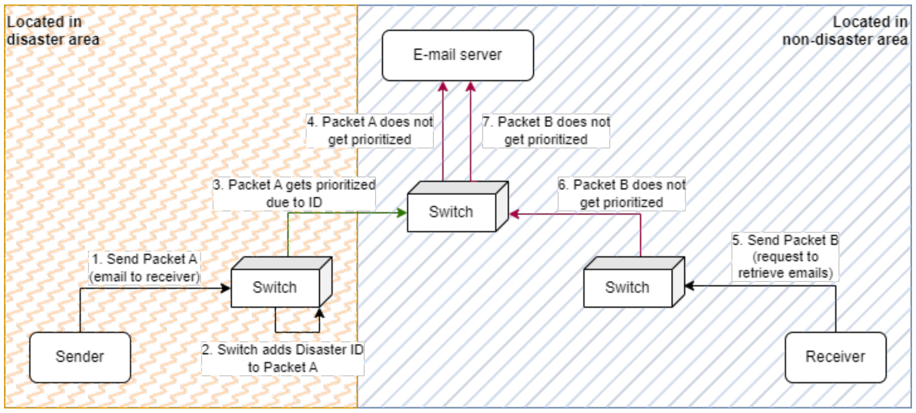

Located in disaster area
Located in non-disaster area
E-mail server
4. Packet A does not get prioritized
7. Packet B does not get prioritized
3. Packet A gets prioritized due to ID
Switch
6. Packet B does not get prioritized
1. Send Packet A (email to receiver)
Switch
Switch
5. Send Packet B (request to retrieve emails)
Sender
2. Switch adds Disaster ID to Packet A
Receiver

Figure 3.3: Example of packets being prioritized depending on the location of the participants. Communications between sender and e-mail server are partially prioritized, while the communication between recipient and e-mail server gets fully prioritized.

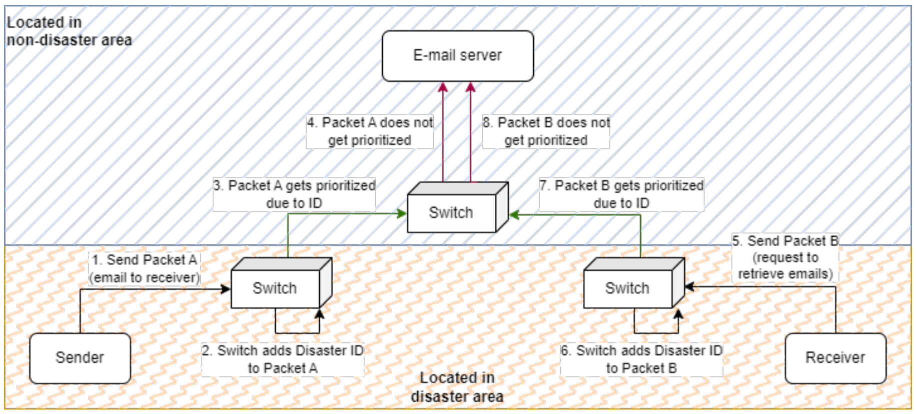

Located in
non-disaster area
E-mail server
4. Packet A does not get prioritized
8. Packet B does not get prioritized
3. Packet A gets prioritized due to ID
Switch
7. Packet B gets prioritized due to ID
5. Send Packet B (request to retrieve emails)
1. Send Packet A (email to receiver)
Switch
Switch
Sender
2. Switch adds Disaster ID to Packet A
Located in
disaster area
6. Switch adds Disaster ID to Packet B
Receiver

Figure 3.4: Example of packets being prioritized depending on the location of the participants. All communications get partially prioritized.

Another argument that could be made is that a network not directly affected by the disaster (i.e. has not experienced physical damage in its components) can still experience challenges to normal operation due to high congestion caused by a raw increase in communications. This network could be geographically close to the disaster, but that is not always the case. Any transit network involved in the communication could be affected by high congestion due to a disaster.

Considering the topology that the Chilean networks have, congestion in transit networks needs to be accounted for in order to achieve a disaster resilient solution. Figure 3.5 shows the Chilean's National University network (Red Universitaria Nacional 2018, REUNA [32]), which shows a simplified view of the national network topology. If the yellow link that directly connects Santiago and Osorno fails, all the traffic usually going through this link will be redirected into the remaining links and nodes inside the white rectangle. These nodes will all become transit networks.

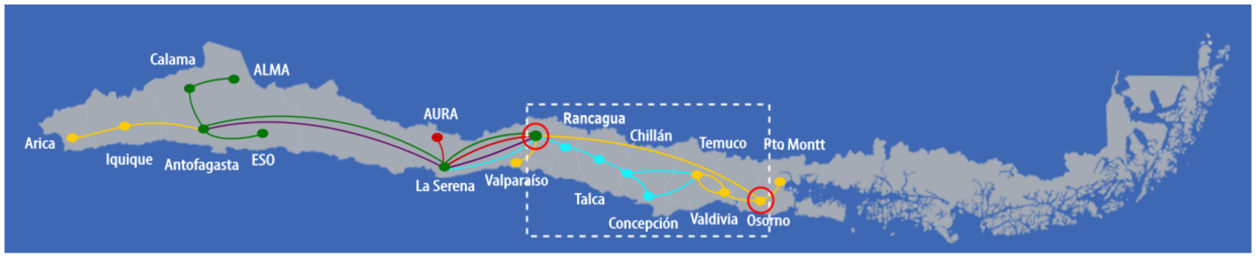


Figure 3.5: Example of nodes becoming transit networks over the Chilean's National University Network (REUNA 2018 [32]). Consider the Santiago and Osorno nodes circled in red. If the yellow link between them fails, all communications between these cities needs to be re-directed between the remaining nodes inside the white rectangle.

This is why the we propose to not only have two methods of operation (disaster and non-disaster) as originally proposed by Ogawa et al. in [42], but to also have a third mode which also prioritizes packets that have a disaster ID attached, but the switches inside a network that works under this mode would not attach disaster IDs to packets coming from (or arriving to) it, as its geolocation was not struck by the disaster. A potential high level condition to trigger this third execution mode could be "high congestion is detected in the monitored network and a disaster occurred recently at a different location".

3.4.3 Network configuration with multiple Internet Service Providers

In Section 2.3.2.10 we see that authors of the reviewed studies designed their SDN solutions based on different assumptions regarding domains and their configurations.

Enforcing the same configuration across different ISPs is not a realistic measure to implement because SDN implies a change in the core architecture of the network. Therefore, for this work, we will assume that the implementation of the solution will be within ISP domains. Specifically, we will assume implementation at the Autonomous System (AS) level.

With this implantation level no assumptions about neighbor domains are made. Either traditional or SDN architecture could be used by the neighbor ASs. Even if they were using SDN as their architecture, they could be using a different SDN configuration.

If a neighbor AS happened to implement the same SDN solution proposed, some features could be designed for the purpose of having a "cooperating mode", but these should be kept as features and not be a requirement for the core functionality of the designed system.

3.5 Implementation and testing

The authors of the studies gathered in this review were specific about listing the tools used to implement and test their proposals.

However, implementation details of their solutions were usually not described. We were not able to find details such as class diagrams nor programming patterns used. Controller applications, one of the core features of the SDN architecture and where most of the solutions should be implemented, were barely described nor discussed. This topic is further reviewed in Section 3.5.3.

Sections 3.5.1 and 3.5.2 aim to first detail the tools chosen for this work. The remaining Sections aim to provide specific details about how to take advantage of the SDN architecture and OpenFlow features in order to design a disaster resilient solution.

3.5.1 Network simulator

The literature review shows in Section 2.3.2.11.5 that Mininet is by far the most used network emulator in the reviewed articles. In addition to this, we found that Mininet is actively developed and supported, it is open source and also it is well documented online [13]. Due to these reasons and because of the ability it provides to experiment with SDNs, we decided to use Mininet as the emulator of choice to create virtual networks for testing purposes.

3.5.2 SDN Controller Platform

Different SDN controller platforms were chosen by the studies of this review: NOX, POX, ONOS, Ryu, OpenDaylight and Floodlight. They all follow and implement the SDN architecture. As reviewed in Section 2.3.2.11.1, the reasoning of choosing one controller over another is not well explored, with the few reasonings provided not dependent on the controller framework specifics such as implementation language, OpenFlow compatibility, etc.

We decided to choose over the controllers found in the literature review due to all being open source and being already tested by experts in the area.

Due to the author's familiarity with the language, using a Python based controller was preferred. Out of the controllers found in the literature review, POX and Ryu meet this preference.

Regarding OpenFlow compatibility, POX only supports OpenFlow 1.0 [2] while Ryu fully supports OpenFlow from versions 1.0 to 1.5 [11]. As key features such as QoS control are only available since OpenFlow version 1.3, Ryu was deemed as the best candidate out of these two controllers.

We continued research and found that Ryu is well documented online [11] and is also compatible with Mininet, the most used network simulator found in the literature review.

Due to these reasons, it was decided to use Ryu as the controller platform for this work's implementation.

3.5.3 SDN Controller Applications

Studies found in the literature review at most mention whether they implemented their algorithms and proposals as SDN controller applications. Other implementation details related to SDN applications are not explained. This is likely because (1) the northbound interface depends on the specific controller framework used, resulting in variations in the specific features provided by the controller for application use; and (2) as the core SDN features remain the same, the reviewed proposals could have been implemented in any controller framework that supports the SDN architecture.

To make the most out of the features provided by the Ryu controller, its API is analized in the next subsection.

3.5.3.1 Ryu Controller Applications

As Ryu was chosen as the controller framework, we summarized the main features of the Ryu Application API below [1]:

- The Ryu application programming model is based on threads, events and event queues.
- Ryu applications send asynchronous events to each other. The controller itself can also send events to the applications and receive events from them.
- Each Ryu application has a FIFO receive queue for events and a thread for event processing. This thread calls the appropriate event handler from the application and blocks it, preventing the processing of new events. We consider this feature important as it eases implementation of applications, but is also aware of a potential bottleneck if the application takes a long time to process the event.
- Ryu provides OpenFlow events classes that support OpenFlow messages and functionalities. The controller emits these events to the Ryu applications to communicate status and changes in the data plane. The controller also receives events to, for example, perform changes in the data plane.
- Ryu events are extensible, allowing creation of new events. These can be used to customize communication between applications.

3.5.4 Data plane management with OpenFlow and controller applications

SDN controller applications are the programmable components that allow dynamic management of the data plane. As explained in Section 2.3.2.1, once a new flow is discovered on the switch, a packet-in message is issued to the controller. The controller then notifies the applications about this new flow.

Applications then may install a new instruction on the switch that emitted the packet-in message to handle this new flow, usually related to routing instructions, QoS rules, etc. This is the most common example of data plane management found in the literature review and it makes sense considering, for example, that routing instructions are specific to the switch.

There is, however, nothing that prevents installing rules on multiple switches simultaneously after, for example, receiving a packet-in message from a single switch. This idea seems useful to the author because it has the potential to improve performance due to the multiple switches not having to request instructions to the controller. Figure 3.6 illustrates this scenario. The performance gains are (1) not having to wait for an RTT (packet-in sent to the controller and response from the controller), (2) not having to wait for the controller application to process the request and (3) reduce the controller load by having less requests to process.

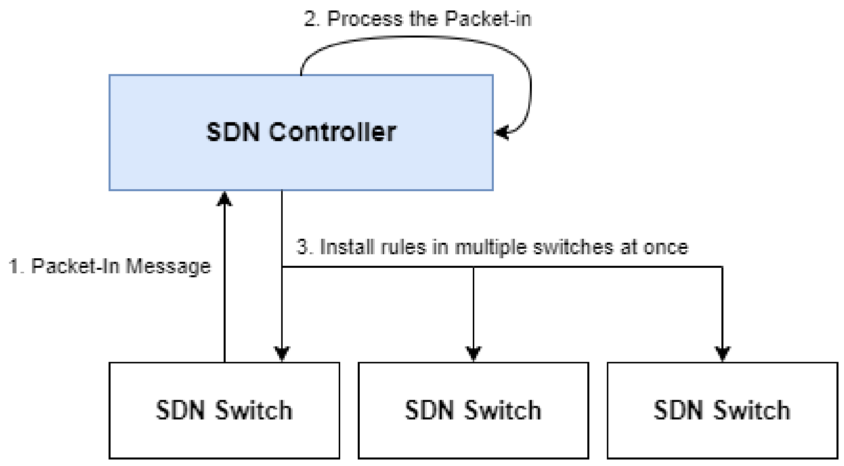


Figure 3.6: SDN Controller installing rules in multiple switches after processing a single Packet-in message.

Now, the next question is which rules can be applied to multiple switches. In the context of this study, rules related to prioritizing packets with disaster IDs can be issued to multiple switches at the beginning of a controller switching to disaster configuration.

Similarly, when switching from disaster configuration to normal execution, removing the prioritizing rules from all switches can be done all at once without having every switch consulting the controller to do so.

The specific rules proposed by in this work that can be applied to multiple switches at once are:

1. OpenFlow rules for ID assignment. If a controller knows that its monitored area is affected by a disaster, it can apply the rule "add a disaster ID to all packets generated in the monitored zone" to all switches controlled in the affected area.
2. OpenFlow rules for prioritization management. OpenFlow meters allow the controller applications to manage maximum bandwidth per flow. Default rules can be applied to all switches as soon as the controller switches from normal operation mode into disaster configuration.

3.5.4.1 OpenFlow's usable features for Prioritization management

One of the key features that OpenFlow provides that is useful for applying QoS measures is the "meter" feature. OpenFlow introduced meters into the protocol for the first time in its version 1.3. Meters allow to control ingress traffic of the switch and guarantee a maximum bandwidth by, for example, dropping packets. This feature is completely manageable through the controller, and thus, through the controller applications.

By installing OpenFlow meter rules in the switches it is possible to ensure QoS for disaster related messages by reserving bandwidth for priority packets.

Adjusting specific bandwidth ratios for priority packets versus normal ones on each switch would still benefit from data analysis on a case by case scenario, because:

1. Different switches have different throughputs.
2. Links connected to output ports of the switch can have different bandwidths.

However, as metering rules do not affect decisions such as which output port to use, the switch does not need to hold a packet until the controller decides on a metering rule. The pre-installed metering rules can be used and the packet can be sent, and if a bandwidth adjustment is needed a new rule can be installed later, which would take effect on subsequent packets of the flow.

Because of the aforementioned reasons is that an SDN solution that aims to build a disaster-resilient WAN must consider OpenFlow meters in their implementation. For this work, it was decided to use meters as the main tool to apply QoS measures.

3.5.5 Conclusion

From the discussion presented in this Chapter, we conclude that SDN can be applied to WANs. There are existing studies that use SDNs to enhance resilience against disaster scenarios, but the methodologies proposed work on isolation and combining techniques is not explored. In addition, there are unanswered questions regarding when and how disaster configuration should be triggered, and how the controller can automatically switch between normal and disaster configurations.

Based on the analysis, we propose designing a disaster-resilient controller that utilizes three levels of ID-based prioritization, each for ECS, disaster-related, and non-disaster-related communications. To make a significant contribution to the field of study, the author should focus on providing answers to the aforementioned questions.

Additionally, the disaster-resilient controller should consider transit networks in its design. It is also important to note that a unified network configuration will not be assumed.

To implement the proposal, Ryu will be used as the controller framework and Mininet will be utilized to simulate a SDN.

Chapter 4

Framework Design

4.1 Introduction

This Chapter describes the framework design, explaining the key functionalities of the controller and the actions it performs according to each situation.

4.2 Controller as a state machine

The controller will be based on the state design pattern. For the framework, three states were defined based on the status of the network managed by the controller.

- Normal operation state: Corresponds to the default state of a controller. This state represents the scenario in which the network managed by the controller operates under normal conditions.
- Disaster In Situ state: Corresponds to a special state of the controller. This state represents the scenario in which the network managed by the controller is geolocated in the same place where a disaster occurred.
- Disaster Ex Situ state: Corresponds to a special state of the controller. This state represents the scenario in which the network managed by the controller is geolocated in a different place than the disaster location.

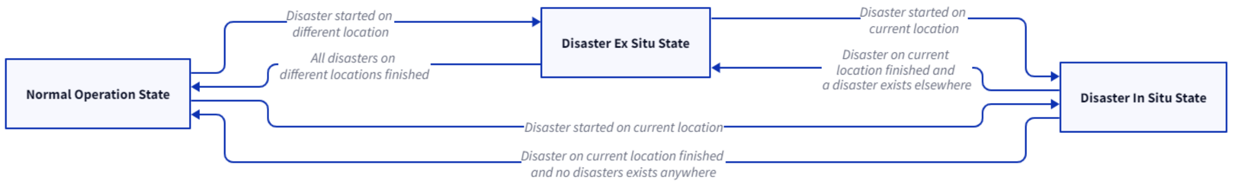
Based on the current state, the controller will perform different actions. These actions are defined in the following Sections of this Chapter.

4.2.1 State change trigger

State changes will be performed upon detecting a combination of the following triggers:

- The beginning or the end of a disaster.
- The location of said disaster.

The diagram in Figure 4.1 shows the state transitions with their specific trigger conditions.



Normal Operation State
Disaster Ex Situ State
Disaster In Situ State
Disaster started on different location
All disasters on different locations finished
Disaster started on current location
Disaster on current location finished and a disaster exists elsewhere
Disaster started on current location
Disaster on current location finished and no disasters exists anywhere

Figure 4.1: State diagram that ilustrates the state transitions of the controller.

All controllers that implement the framework, no matter the current state, will monitor switches and links for aliveness, connections and disconnections that occur on their monitored network. However, the active state determines what actions are performed based on the collected information.

4.3 Network monitoring and notifications

The controller will always monitor the network for failure and recovery of switches and links. The controller's current active state does not interfere with the monitoring. However, the current active state does determine how the information about switches and links is used.

If there is no disaster geolocated in the monitored network, meaning the controller could be either in Normal operation or Disaster Ex Situ state, the controller will analyze disconnections and failures of switches and links and will compare the data with a failure threshold. Once the threshold is reached, the controller switches into Disaster In Situ state, and notifies other controllers about the change.

If a controller in Normal operation state receives the notification that another controller moved into Disaster In Situ, the receiver changes its state into Disaster Ex Situ, meaning that the receiver controller is aware that a disaster struck a network out of its monitoring area.

A controller in a Disaster In Situ state will continue to monitor its network, however this state analyzes new switch connections and link recovery data. Once the network reaches a recovery threshold, the controller transitions into a Normal operation state or a Disaster Ex Situ state, depending on the state of the other remote controllers. A notification will also be sent to other controllers so they register the state transition.

The Disaster In Situ state prevails over the others, meaning that if a controller in this state receives a notification of a different controller moving into Disaster In Situ, the receiver does not perform a state change. It will however, store these notifications so it can check them when exiting the Disaster In Situ state to determine whether it should move into a Normal operation State or into a Disaster Ex Situ state. The former will be picked only if there is no other controller in Disaster In Situ state, and this is determined by reading the notifications previously registered.

4.4 Databases

This Section lists the databases (DB) needed in the framework and how they are updated.

4.4.1 In-network host IP Address DB

Every controller has a DB of IP addresses of hosts directly connected to nodes of the network managed by it.

4.4.2 ECS IP Address DB

The ECS IP Address DB contains IP addresses from registered ECS. This is a pre-constructed DB and it is not expected to dynamically update based on disasters.

4.4.3 Disaster IP Address DB

The Disaster IP Address DB contains IP addresses of hosts geolocated in an area struck by a disaster. Every controller maintains this DB and it associates controller IDs and In-network host DBs. This DB gets updated upon state changes of the controller itself or upon state changes of other controllers.

When the controller moves into a Disaster In situ state, it means the network managed is geolocated in a disaster area, thus it emits a notification to all listening controllers and passes it's controller ID and its In-network host DB.

Every controller that implements this framework (including the remittent controller) reads the notification and adds the received In-network host DB into their Disaster IP Address DB.

When a controller moves out of the Disaster In situ state, it means its hosts are no longer under a disaster scenario, and it emits a "exited Disaster In Situ" notification and passes its controller ID.

Every controller that implements this framework (including the remittent controller) reads the notification and removes the In-network host DB related to the controller ID received.

4.4.4 Disaster In Situ Controller DB

As seen in Section 4.2.1, when a controller exits the Disaster In Situ state it moves to either a Normal operation or Disaster Ex Situ state. To decide which state is picked, each controller maintains a DB which registers other controllers in Disaster In Situ state.

The database gets updated upon receiving notifications of other controllers moving into and exiting a Disaster In Situ state. It stores the IDs of controllers currently in Disaster In situ state. The IDs get removed if that controller exits said state.

4.5 Prioritization IDs

In Section 3.4.2.1 the thesis author proposed to assign IDs to ECS communications, communications coming from or going to a disaster area, and non-disaster related communications. The specific IDs used are defined below:

- ECS ID: Packets emitted from ECS (or directed to them) get this ID attached.
- Disaster ID: Packets emitted from areas affected by the disaster (or directed to them) get this ID attached.
- Non-Disaster ID: Packets that do not match the previously described conditions get this ID attached.

Packets that do not already have an ID should go through an ID assignment process. The Non-Disaster ID exists to mark packets that already went through the process of ID assignment and did not get a high-priority ID, preventing future queries for assignment.

4.6 ID assignment algorithm

The ID assignment process varies depending on the controller state. The general idea behind this statement is that under normal operation there is no need to prioritize, thus there is no need for the IDs. This means that controllers will only attach IDs if they are either in a Disaster In Situ or Disaster Ex Situ state. The algorithm for ID assignment used in Disaster in situ state and Disaster ex situ state ID attachment is described below.

4.6.1 Algorithm for ID attachment

1. If a packet does not have an ID and has a source or destination IP address present on the ECS IP Address DB, the controller attaches an ECS ID to the packet.
2. If the packet did not get an ECS ID, the controller attaches a Disaster ID only if the packet has a source or destination IP address present on the Disaster IP Address DB.
3. If the packet did not get Disaster ID, the controller attaches a Non-Disaster ID.

While the algorithm is the same for Disaster in situ state and Disaster ex situ state, each controller maintains their Disaster IP Address DB differently based on its current state, which leads to different ID assignments.

4.7 Packet prioritization

The prioritization should be active only when disaster scenarios exist, thus the controller states that enable this behavior are Disaster In Situ and Disaster Ex Situ states.

Prioritization will be performed by distributing the available bandwidth based on the priority levels defined in Section 3.4.2.1. Thus, packets with IDs defined in Section 4.5 will be prioritized as follows:

- ECS ID: Highest priority.
- Disaster ID: Second priority.
- Non-Disaster ID: Lowest priority.

Note that packets with ECS ID are expected to be the lowest in quantity, as only a limited number of registered entities emit them. The bulk of packets are the ones containing either Disaster or Non-Disaster IDs.

Bandwidth assigned to ECS ID packets should be the minimum that prevents packet loss. The remaining bandwidth is distributed between Disaster and Non-Disaster ID packets, favoring the former.

Chapter 5

Framework Implementation

5.1 Introduction

This Chapter describes the implementation details of the proposed Disaster Resilient SDN Framework for Chilean networks. As previously discussed in Section 3.5.2, Ryu was chosen as the SDN controller platform, thus the proposed framework uses the Ryu API to manage the network and implement the features and functionalities described in Chapter 4.

5.2 Key technology

5.2.1 SDN Hub Starter Kit

SDN Hub created a VM which contains various software and tools pre installed. It contains Ryu 3.22 which is the version in which this framework got implemented.

5.3 Controller applications

In order to implement all functionalities described in Chapter 4, three controller applications were designed. The applications are:

- Disaster monitor application: Monitors the network and notifications, and sets the controller state accordingly.
- ID attachment application: Attaches ECS IDs, Disaster IDs and Non-Disaster IDs to packets when needed, based on the current state of the controller.
- Prioritization application: Applies QoS policies and manages flow prioritization based on the ID each flow has, based on the current state of the controller.

The design decision of separating functionalities in different applications was taken with encapsulation, maintainability and extensibility principles in mind:

- The disaster detection application works on its own and could be used in any context. Any future new application could read the controller's current state and apply policies based on the disaster situation, allowing the development of new features.
- The ID attachment application and the Prioritization application work closely as they apply and read the same IDs, respectively. However, the Prioritization application does not need to know how an ID was applied, nor the ID attachment application needs to know the mechanism used to prioritize flows.

In order to use the features provided by the Ryu controller framework, each application is implemented as a Ryu application. Every Ryu application is a class that extends the RyuApp class, provided by the Ryu library.

5.3.1 Disaster monitor application

The Disaster monitor application monitors the network and listens to notifications sent from other controllers to set the current status of its controller. It is the core application of the proposed framework as the remaining applications decide which actions should be performed based on the current state of the controller.

The application contains two key objects that allows it to execute its core functionalities:

- Switch counter object
- State object

Their functionalities are described in their class definitions in the following Sections.

5.3.1.1 Switch counter class

The switch counter class is, as the name suggests, a class whose main functionality is to keep track of the number of active switches and the historical maximum number of active switches.

Their methods allow to register connections and disconnections; and get statistics such as the current number of active switches and the active switch ratio, defined as the number of active switches divided by the historical maximum number of switches.

The disaster monitor application has access and makes use of these methods as it contains an instance of this class.

5.3.1.2 State module

The state module corresponds to four classes that follow the State design pattern. The UML class diagram on Figure 5.1 shows the general architecture.

Each state instance has a reference to the Disaster monitor application that contains it so it can send messages to it through its methods.

The behaviors that differentiate each state are listed below:

1. The type of notification that instructs the Disaster monitor application to be sent to the applications upon creating the state. The notification is determined on the constructor of each State.
2. How it processes the switches' information. This behavior is implemented in the switch handler methods.
3. How it processes notifications received from remote controllers regarding their disaster status.

Due to limitations of the testing environment, the notification management between remote controllers were not implemented as they would have not been able to be tested. Thus, remote controller notifications are proposed as future work in Section 7.2.

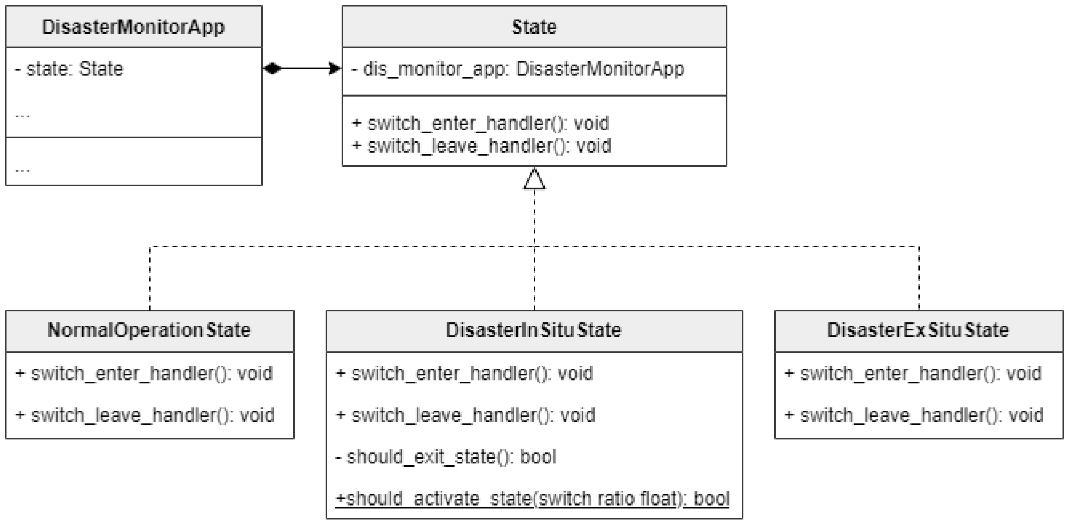

DisasterMonitorApp
- state: State
...
...
State
- dis_monitor_app: DisasterMonitorApp
+ switch_enter_handler(): void
+ switch_leave_handler(): void
NormalOperationState
+ switch_enter_handler(): void
+ switch_leave_handler(): void
DisasterInSituState
+ switch_enter_handler(): void
+ switch_leave_handler(): void
- should_exit_state(): bool
+should_activate_state(switch_ratio float): bool
DisasterExSituState
+ switch_enter_handler(): void
+ switch_leave_handler(): void

Figure 5.1: UML class diagram of the State module.

5.3.2 ID attachment application

The ID attachment application attaches ID to packets based on the controller current state. In particular, it attaches IDs when the controller is in any state different from Normal operation state. The application makes use of the OpenFlow capabilities to attach IDs through the Ryu API.

5.3.2.1 Flow entries management for ID attachment

The application reserves a flow table for itself in each switch of the controller's network, which we call the *ID attachment table*. The application manages this table's flow entries and is able to manipulate packet IDs through them.

When the application starts, it sets a default flow entry to all switches:

◦ Default "Table miss" flow entry: It matches all packets and has the lowest priority (zero). It sends packets directly to the Routing table.

Under a Normal operation state controller, the application does nothing and the *ID attachment table* only forwards packets through the switch pipeline by using the Table miss flow entry. Once the application reads an "exited Normal operation state" event from its controller, it starts modifying the *ID attachment table* so it begins to attach IDs.

The application then creates and sets up three kind of flow entries in the *ID attachment table*:

- Default "Has ID" flow entries: These flow entries match packets that already have their ID assigned, thus it directly sends them to the next table in the pipeline, the Prioritization Table. These flow entries get the highest priority in the table (ten).
- Default "Attach ECS ID" flow entries: These flow entries are created to match flows that have a source or destination address present on the ECS DB. Packets belonging to these flows get an ECS ID attached and are forwarded to the next table in the pipeline, the Prioritization Table. These flow entries get the second highest priority in the table (nine).
- Default "No ID" flow entry: This flow entry matches all packets and is created with the second lowest priority (one) in the table. This flow instructs to send a packet-in message to the controller so this application can decide which ID should be attached to packets belonging to this flow by setting a new "attach" flow.

Note that a packet only reaches the default "No ID" flow entry if it doesn't match any other flow entry with higher priority in the table. With these newly defined flow entries present in the table, the default "table miss" flow entry defined at the beginning is not going to be reached.

It is important to mention that the switch only consults the controller if the first packet of a new flow reaches the "No ID" flow entry. Once the controller receives the packet-in message, the application extracts the source and destination IP address of this packet and checks if any of them are present in the Disaster IP Address DB. With this process the application is able to determine whether a Disaster ID or Non-Disaster ID should be attached. Then, it attaches one of the following flow entries:

- Dynamic "Attach Disaster ID" flow entries: This flow entry matches flows that have a specific source or destination IP address that belongs to a disaster stricken area. Packets belonging to these flows get a Disaster ID attached and are forwarded to the next table in the pipeline, the Prioritization Table. These flow entries get the third highest priority in the table (eight).
- Dynamic "Attach Non-Disaster ID" flow entries: This flow entry matches flows that have a specific source and destination IP addresses and neither of them belongs to a disaster stricken area. Packets belonging to these flows get a Non-Disaster ID attached and are forwarded to the next table in the pipeline, the Prioritization Table. These flow entries get the fourth highest priority in the table (seven).

Once the flow entry is installed on the switch, the switch is going to be able to process packets of the same flow without having to consult the controller again, because these subsequent packets will match with the recently installed dynamic flow entry.

Finally, when the controller transitions into a Normal operation state, the application deletes all flow entries on the *ID attachment table*, with the exception of the default Table miss flow entry.

In summary, in a Normal operation mode the table should only contain the default Table miss flow entry. In any other state, the table will contain multiple flow entries. An example of a populated *ID attachment table* is shown in Figure 5.2.

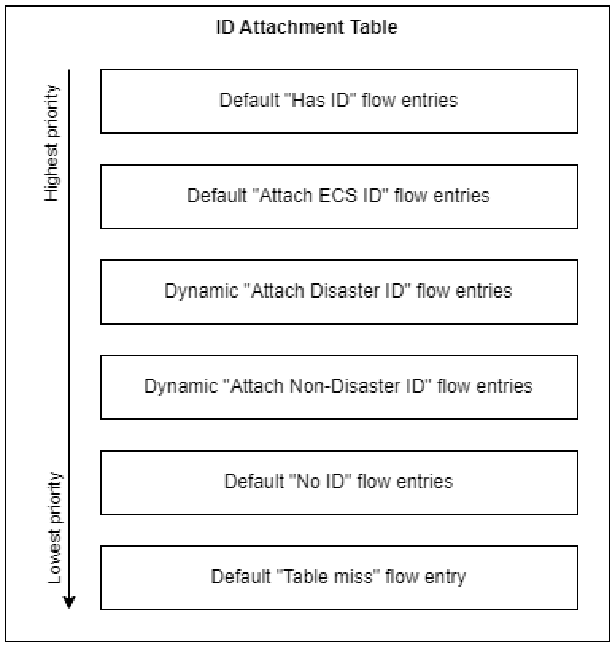

ID Attachment Table
Highest priority
Lowest priority
Default "Has ID" flow entries
Default "Attach ECS ID" flow entries
Dynamic "Attach Disaster ID" flow entries
Dynamic "Attach Non-Disaster ID" flow entries
Default "No ID" flow entries
Default "Table miss" flow entry

Figure 5.2: Example of the ID attachment table. Packet matching against each flow entry is performed based on the priority field of the latter, starting from the highest priority. Once a match is found the process finishes and no other flow entries are evaluated.

5.3.3 Prioritization application

The Prioritization application also acts based on the controller current state, dictated by the Disaster Monitor application. Similarly to the ID Attachment application, the Prioritization application starts its active mode when the controller state is different from the Normal operation state.

The sole purpose of the Prioritization application is to manage bandwidth rates based on the ID each flow has. Note that ID assignment is a problem already solved and handled by the ID Assignment application described in Section 5.3.2.

The Prioritization application makes use of the OpenFlow capabilities through the Ryu API in order to achieve its objective. In particular, it uses the meter features introduced in OpenFlow 1.3. Meters enable OpenFlow to implement rate-limiting as a QoS constraining operation.

5.3.3.1 Meter entries management

Meter are defined on each switch on its Meter table. The application defines three meter entries in the Meter table, one for each prioritization ID defined in Section 4.5. The meter will measure and control the rate of all packets forwarded to it.

5.3.3.2 Flow entries management

In order to perform the packet forwarding process into the Meter table, the prioritization application reserves a flow table from each switch connected to the controller, which we call the *Prioritization table.* The only requirement for this table is that it must have a higher table ID than the ID attachment table, so it is guaranteed that packets will have a prioritization ID once they reach the *Prioritization table.*

When the application starts its active mode, it sets the three default flow entries in the *Prioritization table*, each one matching each prioritization ID defined in Section 4.5 with the instruction to forward packets to the corresponding meter previously defined in the Meter table.

If the packet was not dropped by the meter entry that processed it, the packet continues traversing the switch's OpenFlow pipeline.

5.4 Relation and dependency on other Ryu applications

In order to manage routing over the monitored network, the controller must have a routing application that takes care of this task. For this purpose, the author decided to use a modified version of *L2LearningSwitch*, an application that by default comes with Ryu.

This application manages routing by using one flow table of the pipeline on each switch. By default, it used the first table in the pipeline. For the purpose of this framework, we made it use a flow table with higher ID than the *Prioritization table* from the Prioritization application. We will refer to this table as the *Routing table.*

Another useful application that comes with Ryu is the *Switches* application, which maintains an up-to-date view of the switches in the network. The ID attachment application and the Prioritization application both request switch data to the Switches application so they can install their default flows on them.

These two applications, L2LearningSwitch and Switches, could be replaced by any other application that performs the same tasks.

5.5 Disaster resilient controller within the SDN architecture

Having delineated the applications and modules of the disaster resilient controller, we can now show how the controller operates within the SDN architecture.

Figure 5.3 depicts the SDN architecture, showcasing the applications presented in this Chapter and their relationships with the implemented modules.

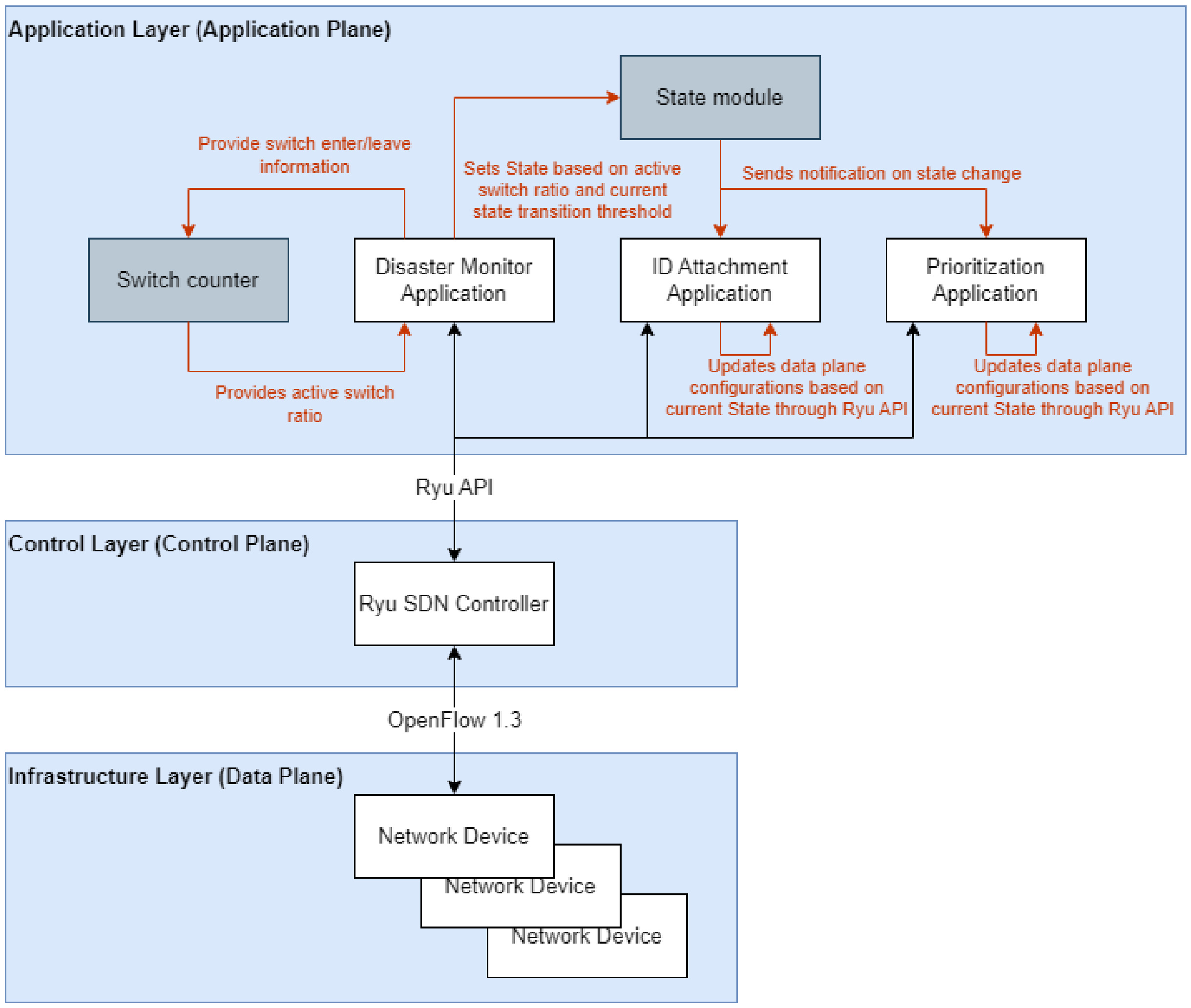

Application Layer (Application Plane)
State module
Provide switch enter/leave information
Sets State based on active switch ratio and current state transition threshold
Sends notification on state change
Switch counter
Disaster Monitor Application
ID Attachment Application
Prioritization Application
Provides active switch ratio
Updates data plane configurations based on current State through Ryu API
Updates data plane configurations based on current State through Ryu API
Ryu API
Control Layer (Control Plane)
Ryu SDN Controller
OpenFlow 1.3
Infrastructure Layer (Data Plane)
Network Device
Network Device
Network Device

Figure 5.3: Implemented disaster resilient controller within the SDN archietcture.

5.6 Source code

The source code of the implemented disaster resilient controller can be found on the repository [43].

Chapter 6

Testing and Results

6.1 Introduction

The testing and results Chapter provides an overview of the evaluation of the disaster resilient controller with its three designed applications in the context of a Mininet and Open vSwitch (OVS) testbed. The Chapter lists the testing methodologies, including simple and complex scenarios for both functional and non-functional testing, with the latter focusing on simulating a disaster scenario over a network. Limitations and challenges were found, which are also discussed. Overall, this Chapter presents the testing details and the results obtained from the experimentation with the designed applications and the controller in a simulated SDN environment.

6.2 Key technology

Similarly to the framework implementation, for testing we again used the SDN Hub VM which, by default, has the following software installed that allows to create a simulated network for testing:

- Mininet 2.2.1: Network simulator
- Open VSwitch 2.3: Open source virtual switch that supports OpenFlow.
- iPerf 2.0.5: Tool for traffic generation and bandwidth measurements.

However, while Open VSwitch 2.3 supports Openflow 1.3 messages, it is not able to process all of them. In particular, the support for OpenFlow Meters in Linux datapath was released in version 2.10.0 in 2018, meaning a newer version needed to be installed.

The chosen version of Open VSwitch was 2.17.3 as at the time of building the testing platform, was the most recent release from the long term support (LTS) series.

These tools, in addition with Ryu 3.22 which also comes with the SDN Hub VM, allow to create a testing environment capable of deploying SDN and OpenFlow features.

6.3 Setup

The testing setup requires configuring the network topology through Mininet, configuring the switches through Open VSwitch and starting the Ryu controller. For all these steps, the thesis author programmed the scripts detailed in this Section to automate the setup. We will refer to these three scripts as the *Setup scripts*.

6.3.1 Simple topology builder script

A simple topology can be built using the script *set_ mininet.sh*. It invokes a Mininet command to create the desired topology, determine the type of the switches and set the controller type and its default IP address. Setting the IP is critical so the switches can connect to the controller.

6.3.2 Switch configuration script

After the topology is built, we need to configure every switch in the network to use the correct OpenFlow version and the datapath type. This configuration is needed to support meters. To achieve this, we invoke Open VSwitch's commands on each switch with the script *set_ bridges.sh*.

6.3.3 Ryu script

The sole purpose of this script is to launch the Ryu controller with the required applications. This switch does not depend on the aforementioned scripts.

The script *minimal_ ryu.sh* launches a Ryu command to start the controller with the desired applications. The applications launched are the ones proposed in this work:

- Disaster monitor application
- ID attachment application
- Prioritization application

The other applications mentioned in Section 5.4 are coded as a requirement for the latter two, thus these get instantiated automatically by them and there is no need for them to be explicitly called on launch.

6.4 Test scenarios

This Section is divided into functional testing and non-functional testing, with the latter focusing on simulating a disaster scenario.

6.4.1 Functional testing

To test the basic functionalities of the developed framework, a small topology is picked to ease the analysis. The chosen topology can be seen in Figure 6.1, which corresponds to a linear topology of four switches where each of them has one host connected.

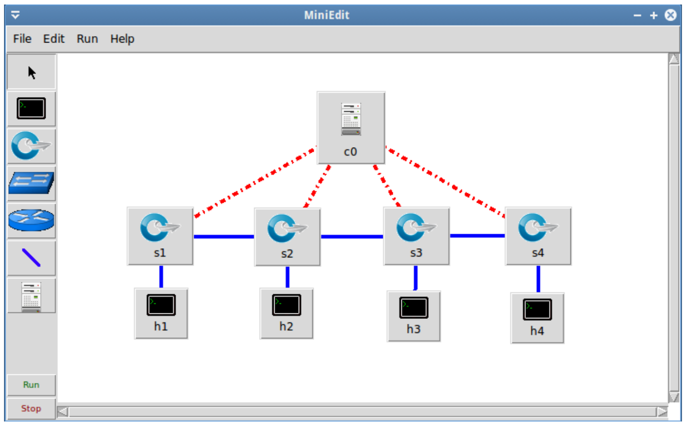
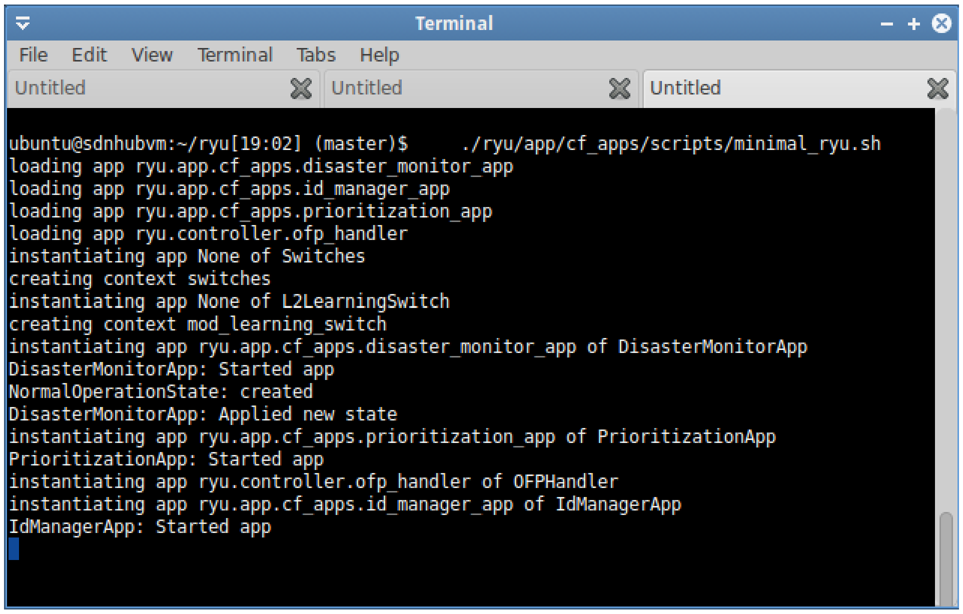


Figure 6.1: Linear topology with four switches and hosts, used for functional testing. Visualized through Miniedit, a GUI editor for Mininet.

6.4.1.1 Starting the controller

If everything was configured correctly, the controller should be able to successfully start and load the applications. Simply running the Ryu script is enough.

Figure 6.2 shows the log emitted by the controller upon a successful start. It shows all the applications loaded into it.

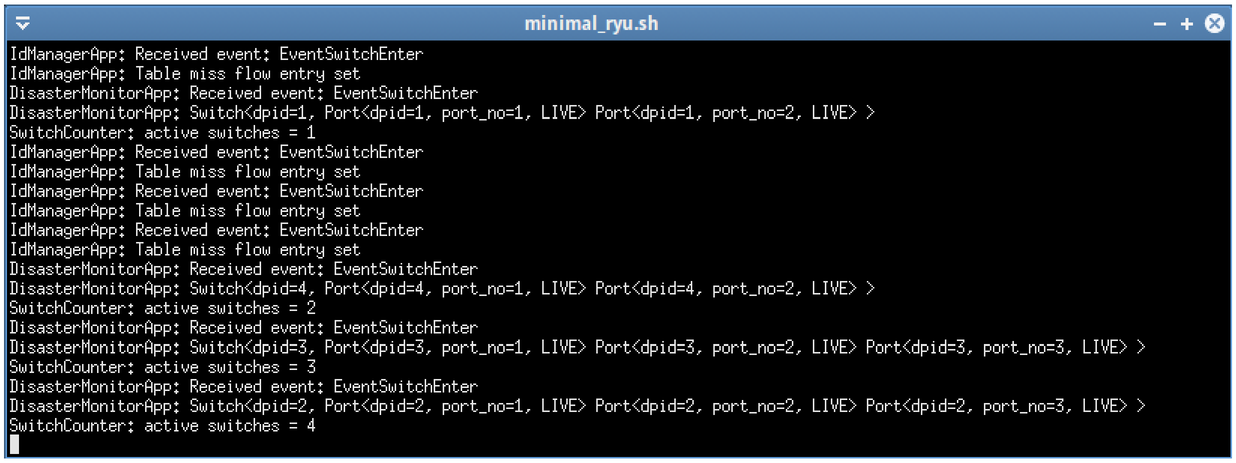


```
ubuntu@sdnhubvm:~/ryu[19:02] (master)$     ./ryu/app/cf_apps/scripts/minimal_ryu.sh
loading app ryu.app.cf_apps.disaster_monitor_app
loading app ryu.app.cf_apps.id_manager_app
loading app ryu.app.cf_apps.prioritization_app
loading app ryu.controller.ofp_handler
instantiating app None of Switches
creating context switches
instantiating app None of L2LearningSwitch
creating context mod_learning_switch
instantiating app ryu.app.cf_apps.disaster_monitor_app of DisasterMonitorApp
DisasterMonitorApp: Started app
NormalOperationState: created
DisasterMonitorApp: Applied new state
instantiating app ryu.app.cf_apps.prioritization_app of PrioritizationApp
PrioritizationApp: Started app
instantiating app ryu.controller.ofp_handler of OFPHandler
instantiating app ryu.app.cf_apps.id_manager_app of IdManagerApp
IdManagerApp: Started app
```

Figure 6.2: Log emitted by the controller upon a successful start.

6.4.1.2 Switch discovery

Mininet sets the controller IP onto the switches of the topology. Thus, the controller should be able to discover the switches automatically after being started, i.e. after executing the setup scripts (Topology builder, Switch configuration and Ryu scripts). Figure 6.3 shows the controller discovering the switches.

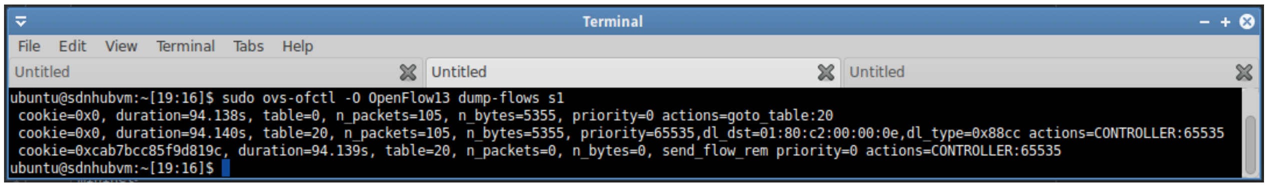


```
IdManagerApp: Received event: EventSwitchEnter
IdManagerApp: Table miss flow entry set
DisasterMonitorApp: Received event: EventSwitchEnter
DisasterMonitorApp: Switch<dpid=1, Port<dpid=1, port_no=1, LIVE> Port<dpid=1, port_no=2, LIVE> >
SwitchCounter: active switches = 1
IdManagerApp: Received event: EventSwitchEnter
IdManagerApp: Table miss flow entry set
IdManagerApp: Received event: EventSwitchEnter
IdManagerApp: Table miss flow entry set
IdManagerApp: Received event: EventSwitchEnter
IdManagerApp: Table miss flow entry set
DisasterMonitorApp: Received event: EventSwitchEnter
DisasterMonitorApp: Switch<dpid=4, Port<dpid=4, port_no=1, LIVE> Port<dpid=4, port_no=2, LIVE> >
SwitchCounter: active switches = 2
DisasterMonitorApp: Received event: EventSwitchEnter
DisasterMonitorApp: Switch<dpid=3, Port<dpid=3, port_no=1, LIVE> Port<dpid=3, port_no=2, LIVE> Port<dpid=3, port_no=3, LIVE> >
SwitchCounter: active switches = 3
DisasterMonitorApp: Received event: EventSwitchEnter
DisasterMonitorApp: Switch<dpid=2, Port<dpid=2, port_no=1, LIVE> Port<dpid=2, port_no=2, LIVE> Port<dpid=2, port_no=3, LIVE> >
SwitchCounter: active switches = 4
```

Figure 6.3: Log emitted by the controller when it discovers new switches.

In addition to discovering the switches, the default flow entries should be installed in the flow tables. These can be seen in Figure 6.4. As described in Section 5.3.2, the ID attachment application installs its default table miss flow entry on the *ID attachment table* (table zero), which directs all packets into the *Routing table* (table twenty, which is managed by the Routing application).

From these results, we conclude that the Mininet configuration over the virtual switches allows them to communicate with the controller. The controller is also able to register them and to correctly apply the default flows required by the different applications.



```
ubuntu@sdnhubvm:~[19:16]$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1
 cookie=0x0, duration=94.138s, table=0, n_packets=105, n_bytes=5355, priority=0 actions=goto_table:20
 cookie=0x0, duration=94.140s, table=20, n_packets=105, n_bytes=5355, priority=65535,dl_dst=01:80:c2:00:00:0e,dl_type=0x88cc actions=CONTROLLER:65535
 cookie=0xcab7bcc85f9d819c, duration=94.139s, table=20, n_packets=0, n_bytes=0, send_flow_rem priority=0 actions=CONTROLLER:65535
ubuntu@sdnhubvm:~[19:16]$
```

Figure 6.4: Use of Open VSwitch comand to see the current flow entries installed on switch "s1". The flow entries shown correspond to the default ones installed on said switch upon it being discovered by the controller.

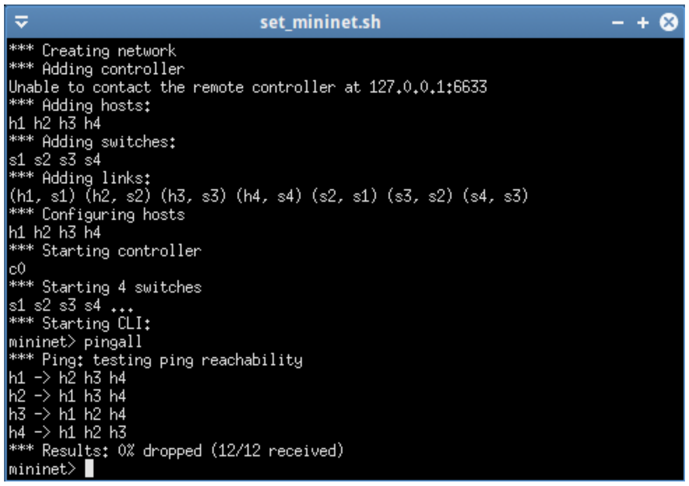
6.4.1.3 Routing test

As the applications proposed by the author manipulate flow tables, it is important to confirm that the routing capabilities of the controller are not disrupted by the flow table management of said applications.

We again execute the three setup scripts for this experiment, then we perform the *pingall* command in Mininet to test the connectivity of all four switches in the network. The purpose of this test is to confirm that the necessary flow entries for routing get installed and that every host is able to communicate with all destinations.

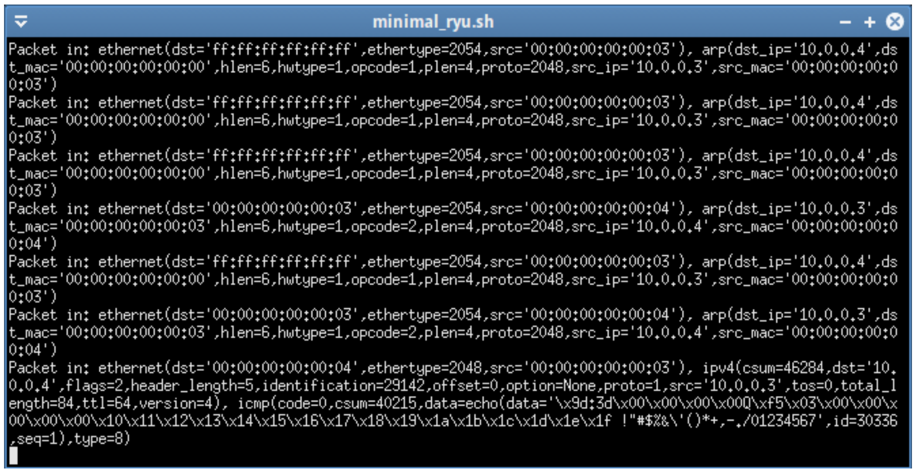
Figure 6.5 shows the *pingall* command executed in Mininet and the connectivity results, which are successful. Figure 6.6 shows the packet-in messages the controller received which triggered the installment of flow entries into the switches. Finally, Figure 6.7 shows one of the switches with its respective new flows. In this case we can see the same existing flows as shown in the Switch discovery experiment and the new routing rules that allowed all switches to be reachable.

We conclude from this result that the routing application used (L2LearningSwitch) provided by Ryu is capable of performing simple network routing discovery and correctly applies flow entries that allow packet forwarding. The results also show that the *ID attachment table*, the *Prioritization table* and the *Routing table* work together and do not disrupt packet forwarding.



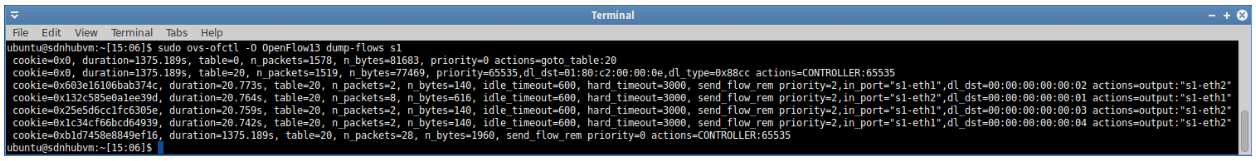
```
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3 s4
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (h4, s4) (s2, s1) (s3, s2) (s4, s3)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 4 switches
s1 s2 s3 s4 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
mininet>
```

Figure 6.5: Use of Mininet command *pingall*. Full connectivity is achieved due to the controller correctly installing routing flows.



```
Packet in: ethernet(dst='ff:ff:ff:ff:ff:ff',ethertype=2054,src='00:00:00:00:00:03'), arp(dst_ip='10.0.0.4',ds
t_mac='00:00:00:00:00:00',hlen=6,hwtype=1,opcode=1,plen=4,proto=2048,src_ip='10.0.0.3',src_mac='00:00:00:00:0
0:03')
Packet in: ethernet(dst='ff:ff:ff:ff:ff:ff',ethertype=2054,src='00:00:00:00:00:03'), arp(dst_ip='10.0.0.4',ds
t_mac='00:00:00:00:00:00',hlen=6,hwtype=1,opcode=1,plen=4,proto=2048,src_ip='10.0.0.3',src_mac='00:00:00:00:0
0:03')
Packet in: ethernet(dst='ff:ff:ff:ff:ff:ff',ethertype=2054,src='00:00:00:00:00:03'), arp(dst_ip='10.0.0.4',ds
t_mac='00:00:00:00:00:00',hlen=6,hwtype=1,opcode=1,plen=4,proto=2048,src_ip='10.0.0.3',src_mac='00:00:00:00:0
0:03')
Packet in: ethernet(dst='00:00:00:00:00:03',ethertype=2054,src='00:00:00:00:00:04'), arp(dst_ip='10.0.0.3',ds
t_mac='00:00:00:00:00:03',hlen=6,hwtype=1,opcode=2,plen=4,proto=2048,src_ip='10.0.0.4',src_mac='00:00:00:00:0
0:04')
Packet in: ethernet(dst='ff:ff:ff:ff:ff:ff',ethertype=2054,src='00:00:00:00:00:03'), arp(dst_ip='10.0.0.4',ds
t_mac='00:00:00:00:00:00',hlen=6,hwtype=1,opcode=1,plen=4,proto=2048,src_ip='10.0.0.3',src_mac='00:00:00:00:0
0:03')
Packet in: ethernet(dst='00:00:00:00:00:03',ethertype=2054,src='00:00:00:00:00:04'), arp(dst_ip='10.0.0.3',ds
t_mac='00:00:00:00:00:03',hlen=6,hwtype=1,opcode=2,plen=4,proto=2048,src_ip='10.0.0.4',src_mac='00:00:00:00:0
0:04')
Packet in: ethernet(dst='00:00:00:00:00:04',ethertype=2048,src='00:00:00:00:00:03'), ipv4(csum=46284,dst='10.
0.0.4',flags=2,header_length=5,identification=29142,offset=0,option=None,proto=1,src='10.0.0.3',tos=0,total_l
ength=84,ttl=64,version=4), icmp(code=0,csum=40215,data=echo(data='\x9d:3d\x00\x00\x00\x00Q\xf5\x03\x00\x00\x
00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./01234567',id=30336
,seq=1),type=8)
```

Figure 6.6: Controller log that shows the reception of packet-in messages sent by the switches.



```
ubuntu@sdnhubvm:~[15:06]$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1
 cookie=0x0, duration=1375.189s, table=0, n_packets=1578, n_bytes=81683, priority=0 actions=goto_table:20
 cookie=0x0, duration=1375.189s, table=20, n_packets=1519, n_bytes=77469, priority=65535,dl_dst=01:80:c2:00:00:0e,dl_type=0x88cc actions=CONTROLLER:65535
 cookie=0x603e16106bab374c, duration=20.773s, table=20, n_packets=2, n_bytes=140, idle_timeout=600, hard_timeout=3000, send_flow_rem priority=2,in_port="s1-eth1",dl_dst=00:00:00:00:00:02 actions=output:"s1-eth2"
 cookie=0x132c585e0a1ee39d, duration=20.764s, table=20, n_packets=8, n_bytes=616, idle_timeout=600, hard_timeout=3000, send_flow_rem priority=2,in_port="s1-eth2",dl_dst=00:00:00:00:00:01 actions=output:"s1-eth1"
 cookie=0x25e5d6cc1fc6305e, duration=20.759s, table=20, n_packets=2, n_bytes=140, idle_timeout=600, hard_timeout=3000, send_flow_rem priority=2,in_port="s1-eth1",dl_dst=00:00:00:00:00:03 actions=output:"s1-eth2"
 cookie=0x1c34cf66bcd64939, duration=20.742s, table=20, n_packets=2, n_bytes=140, idle_timeout=600, hard_timeout=3000, send_flow_rem priority=2,in_port="s1-eth1",dl_dst=00:00:00:00:00:04 actions=output:"s1-eth2"
 cookie=0xb1d7458e8849ef16, duration=1375.189s, table=20, n_packets=28, n_bytes=1960, send_flow_rem priority=0 actions=CONTROLLER:65535
ubuntu@sdnhubvm:~[15:06]$
```

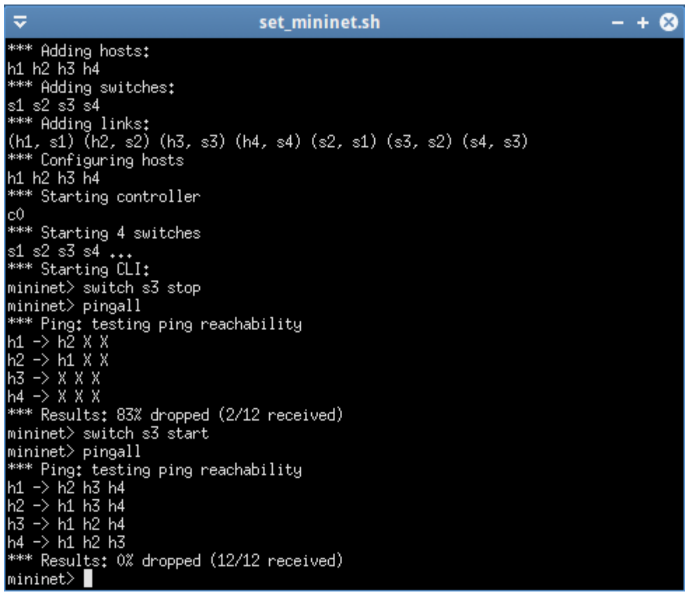
Figure 6.7: Use of Open VSwitch comand to see the current flow entries installed on switch *s1*. The flow entries shown correspond to the default ones installed on said switch upon it being discovered by the controller and the new flow entries added due to routing management.

6.4.1.4 Switch failure and recovery

We now want to test the monitoring capabilities of the controller, in particular, being able to identify a switch that is longer responsive.

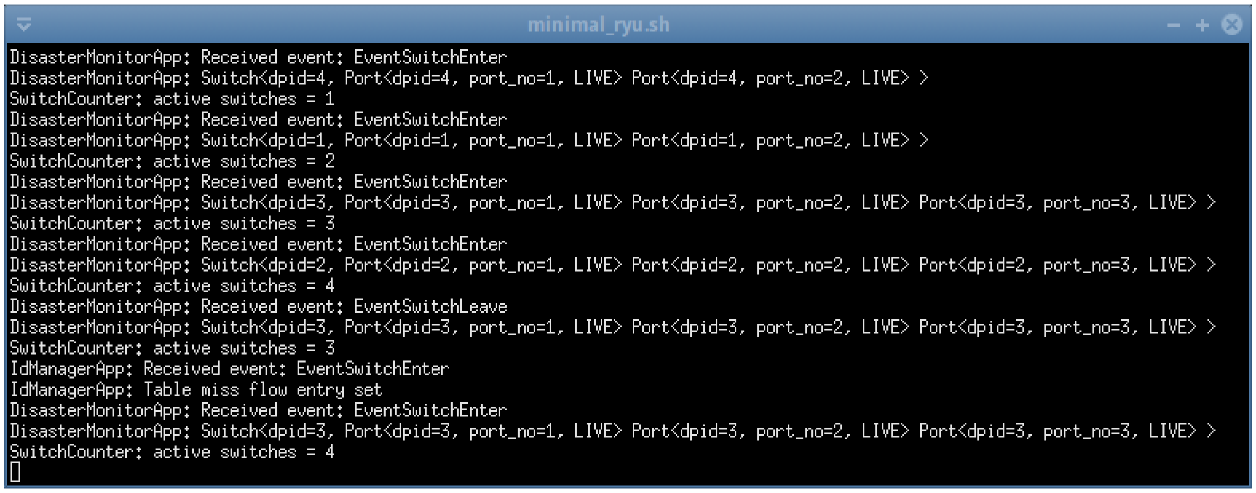
We again execute the three setup scripts for this experiment, thus all four switches are discovered by the controller. We then execute the Mininet command *switch s3 stop* to turn the switch off and then execute the Mininet command *pingall* to confirm that the switch *s3* stopped working. Figure 6.8 shows Mininet's console and the expected result of only *h1* and *h2* hosts being reachable between themselves, because *s3* going down means that now the *s4* switch is now isolated. Figure 6.9 shows the controller point of view, which recognizes the switch with datapath ID three (dpid=3) being no longer responsive.

We then re-connect the switch *s3* by executing the Mininet command *switch s3 start* and again perform the *pingall* command. Figure 6.8 shows that now all nodes are reachable, confirming that *s3* is now up. Figure 6.9 shows the controller successfully capturing the reconnection of the switch.



```
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3 s4
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (h4, s4) (s2, s1) (s3, s2) (s4, s3)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 4 switches
s1 s2 s3 s4 ...
*** Starting CLI:
mininet> switch s3 stop
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 X X
h2 -> h1 X X
h3 -> X X X
h4 -> X X X
*** Results: 83% dropped (2/12 received)
mininet> switch s3 start
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
mininet> 
```

Figure 6.8: Mininet log which shows the experiment of a switch failure and recovery. Switch *s3* is stopped by the command *switch s3 stop*, which isolates switch *s4*. After re-connecting *s3* with the command *switch s3 start*, the connectivity is recovered.



```
DisasterMonitorApp: Received event: EventSwitchEnter
DisasterMonitorApp: Switch<dpid=4, Port<dpid=4, port_no=1, LIVE> Port<dpid=4, port_no=2, LIVE> >
SwitchCounter: active switches = 1
DisasterMonitorApp: Received event: EventSwitchEnter
DisasterMonitorApp: Switch<dpid=1, Port<dpid=1, port_no=1, LIVE> Port<dpid=1, port_no=2, LIVE> >
SwitchCounter: active switches = 2
DisasterMonitorApp: Received event: EventSwitchEnter
DisasterMonitorApp: Switch<dpid=3, Port<dpid=3, port_no=1, LIVE> Port<dpid=3, port_no=2, LIVE> Port<dpid=3, port_no=3, LIVE> >
SwitchCounter: active switches = 3
DisasterMonitorApp: Received event: EventSwitchEnter
DisasterMonitorApp: Switch<dpid=2, Port<dpid=2, port_no=1, LIVE> Port<dpid=2, port_no=2, LIVE> Port<dpid=2, port_no=3, LIVE> >
SwitchCounter: active switches = 4
DisasterMonitorApp: Received event: EventSwitchLeave
DisasterMonitorApp: Switch<dpid=3, Port<dpid=3, port_no=1, LIVE> Port<dpid=3, port_no=2, LIVE> Port<dpid=3, port_no=3, LIVE> >
SwitchCounter: active switches = 3
IdManagerApp: Received event: EventSwitchEnter
IdManagerApp: Table miss flow entry set
DisasterMonitorApp: Received event: EventSwitchEnter
DisasterMonitorApp: Switch<dpid=3, Port<dpid=3, port_no=1, LIVE> Port<dpid=3, port_no=2, LIVE> Port<dpid=3, port_no=3, LIVE> >
SwitchCounter: active switches = 4
```

Figure 6.9: Controller log during the switch failure and recovery experiment. Initially all four switches are up. Then, switch *s3* gets disconnected which is captured by the controller. Once *s3* reconnects, the controller recognizes it. Note that the controller recognized it is the same switch that was disconnected and reconnected due to its datapath ID (dpid).

6.4.2 Simulating a disaster scenario

In the previous Section it is shown that the network simulator setup allows testing of the basic functional capabilities of the designed SDN disaster-resilient controller. We now want to simulate disaster scenarios to validate the correctness and effectiveness of the controller when it applies its designed features.

6.4.2.1 Multiple switch failure and recovery

This experiment is similar to the one described in Section 6.4.1.4, but here it is scaled up to test multiple switch failures and to trigger status changes in the controller.

For this experiment, the controller is configured with an *enter disaster* threshold of 70% active switches, and an *exit disaster* threshold of 80% active switches. The chosen topology corresponds to a tree topology that contains 127 switches. The controller starts as default, meaning it starts in a Normal Operation state.

To test both thresholds, switches get disconnected and reconnected in four steps. Each step and its purpose are listed below:

1. Step A: This experiment disconnects switches without reaching the enter disaster threshold, meaning the Normal Operation state should be maintained.
2. Step B: We continue to disconnect switches and the active switch ratio goes below the enter disaster threshold, thus the controller's state should change into Disaster In Situ.
3. Step C: Now switches start re-connecting, however the exit disaster threshold is not met yet, thus the Disaster In Situ state should be maintained.
4. Step D: More switches reconnect and the active switch ratio goes above the exit disaster threshold, thus the controller should go back into Normal Operation state.

The results of the multiple switch failure experiment are summarized in Table 6.1 for steps A and B. Table 6.2 shows the results of the multiple switch recovery experiment for steps C and D.

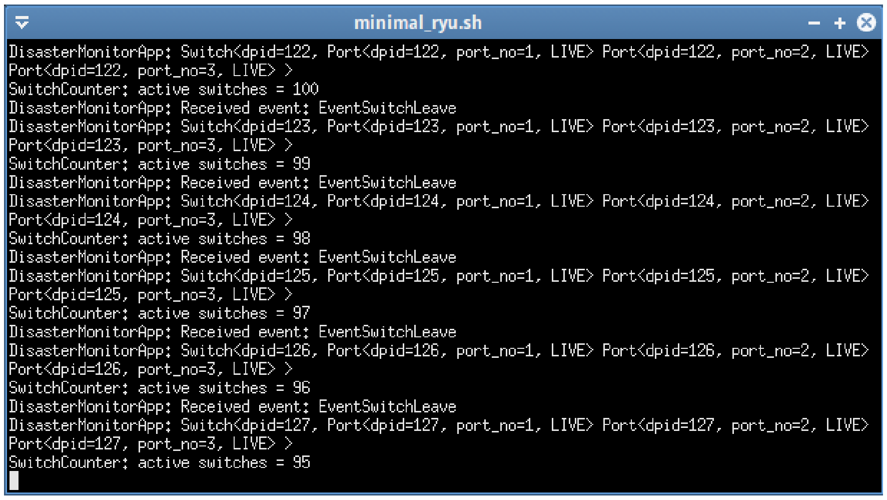
In Figure 6.10 we see the controller log generated when processing steps A and B. Similarly, Figure 6.11 shows the controller log generated when processing steps C and D. The controller logs from step B and C also show that, when the Disaster In Situ state is active, it installs the required flow entries for ID assignment and prioritization. The logs from step B and D also show that the state transitions take place immediately after the switch counter crosses the respective threshold (meaning 88 active switches in step B and 102 active switches in step D).

Initial state	Step	Number of switches in the network	Number of active switches post disaster	% of active switches post disaster	Should move into Disaster In Situ state?	Moved into Disaster In Situ state?
Normal Operation	A	127	95	74.8	No	No
	B	127	85	66.9	Yes	Yes

Table 6.1: Results of the multiple switch failure experiment for steps A and B.

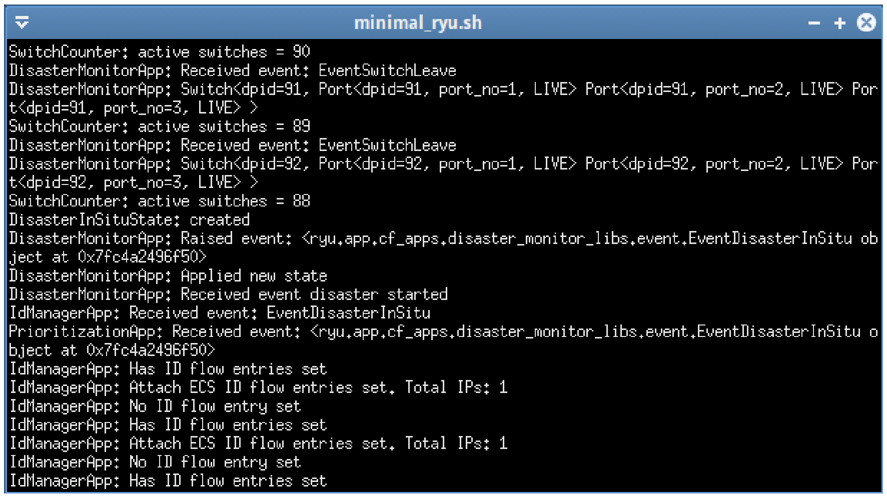
Initial state	Step	Number of switches in the network	Number of active switches post disaster	Number of active switches after recovery	% of active switches post recovery	Should move into Normal Operation state?	Moved into Normal Operation state?
Disaster	C	127	85	95	74.8	No	No
In Situ	D	127	95	110	86.6	Yes	Yes

Table 6.2: Results of the multiple switch recovery experiment for steps C and D.


minimal_ryu.sh

```
DisasterMonitorApp: Switch<dpid=122, Port<dpid=122, port_no=1, LIVE> Port<dpid=122, port_no=2, LIVE>
Port<dpid=122, port_no=3, LIVE> >
SwitchCounter: active switches = 100
DisasterMonitorApp: Received event: EventSwitchLeave
DisasterMonitorApp: Switch<dpid=123, Port<dpid=123, port_no=1, LIVE> Port<dpid=123, port_no=2, LIVE>
Port<dpid=123, port_no=3, LIVE> >
SwitchCounter: active switches = 99
DisasterMonitorApp: Received event: EventSwitchLeave
DisasterMonitorApp: Switch<dpid=124, Port<dpid=124, port_no=1, LIVE> Port<dpid=124, port_no=2, LIVE>
Port<dpid=124, port_no=3, LIVE> >
SwitchCounter: active switches = 98
DisasterMonitorApp: Received event: EventSwitchLeave
DisasterMonitorApp: Switch<dpid=125, Port<dpid=125, port_no=1, LIVE> Port<dpid=125, port_no=2, LIVE>
Port<dpid=125, port_no=3, LIVE> >
SwitchCounter: active switches = 97
DisasterMonitorApp: Received event: EventSwitchLeave
DisasterMonitorApp: Switch<dpid=126, Port<dpid=126, port_no=1, LIVE> Port<dpid=126, port_no=2, LIVE>
Port<dpid=126, port_no=3, LIVE> >
SwitchCounter: active switches = 96
DisasterMonitorApp: Received event: EventSwitchLeave
DisasterMonitorApp: Switch<dpid=127, Port<dpid=127, port_no=1, LIVE> Port<dpid=127, port_no=2, LIVE>
Port<dpid=127, port_no=3, LIVE> >
SwitchCounter: active switches = 95
```

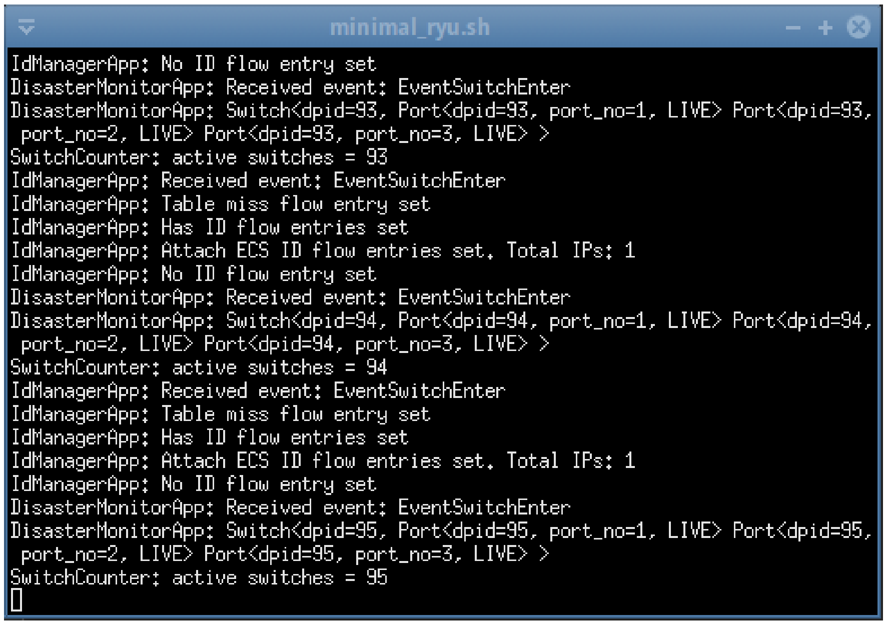
(a) Step A. The controller log shows that the disconnections are captured, but no state transition is performed.


minimal_ryu.sh

```
SwitchCounter: active switches = 90
DisasterMonitorApp: Received event: EventSwitchLeave
DisasterMonitorApp: Switch<dpid=91, Port<dpid=91, port_no=1, LIVE> Port<dpid=91, port_no=2, LIVE> Por
t<dpid=91, port_no=3, LIVE> >
SwitchCounter: active switches = 89
DisasterMonitorApp: Received event: EventSwitchLeave
DisasterMonitorApp: Switch<dpid=92, Port<dpid=92, port_no=1, LIVE> Port<dpid=92, port_no=2, LIVE> Por
t<dpid=92, port_no=3, LIVE> >
SwitchCounter: active switches = 88
DisasterInSituState: created
DisasterMonitorApp: Raised event: <ryu.app.cf_apps.disaster_monitor_libs.event.EventDisasterInSitu ob
ject at 0x7fc4a2496f50>
DisasterMonitorApp: Applied new state
DisasterMonitorApp: Received event disaster started
IdManagerApp: Received event: EventDisasterInSitu
PrioritizationApp: Received event: <ryu.app.cf_apps.disaster_monitor_libs.event.EventDisasterInSitu o
bject at 0x7fc4a2496f50>
IdManagerApp: Has ID flow entries set
IdManagerApp: Attach ECS ID flow entries set. Total IPs: 1
IdManagerApp: No ID flow entry set
IdManagerApp: Has ID flow entries set
IdManagerApp: Attach ECS ID flow entries set. Total IPs: 1
IdManagerApp: No ID flow entry set
IdManagerApp: Has ID flow entries set
```

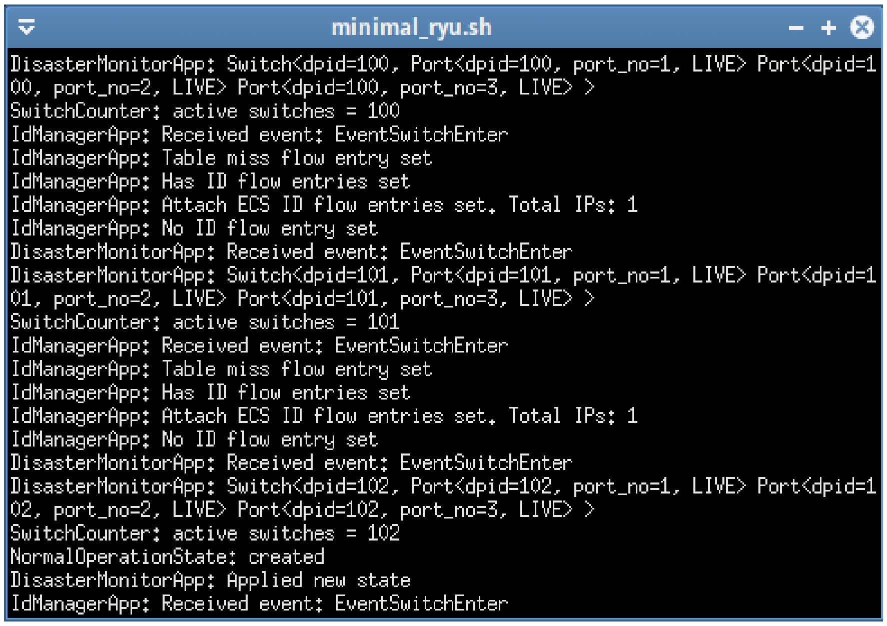
(b) Step B. The controller log shows that the *enter disaster* threshold is reached and the state transition is performed.

Figure 6.10: Controller log generated when processing steps A and B of the multiple switch failure test.



```
IdManagerApp: No ID flow entry set
DisasterMonitorApp: Received event: EventSwitchEnter
DisasterMonitorApp: Switch<dpid=93, Port<dpid=93, port_no=1, LIVE> Port<dpid=93,
 port_no=2, LIVE> Port<dpid=93, port_no=3, LIVE> >
SwitchCounter: active switches = 93
IdManagerApp: Received event: EventSwitchEnter
IdManagerApp: Table miss flow entry set
IdManagerApp: Has ID flow entries set
IdManagerApp: Attach ECS ID flow entries set. Total IPs: 1
IdManagerApp: No ID flow entry set
DisasterMonitorApp: Received event: EventSwitchEnter
DisasterMonitorApp: Switch<dpid=94, Port<dpid=94, port_no=1, LIVE> Port<dpid=94,
 port_no=2, LIVE> Port<dpid=94, port_no=3, LIVE> >
SwitchCounter: active switches = 94
IdManagerApp: Received event: EventSwitchEnter
IdManagerApp: Table miss flow entry set
IdManagerApp: Has ID flow entries set
IdManagerApp: Attach ECS ID flow entries set. Total IPs: 1
IdManagerApp: No ID flow entry set
DisasterMonitorApp: Received event: EventSwitchEnter
DisasterMonitorApp: Switch<dpid=95, Port<dpid=95, port_no=1, LIVE> Port<dpid=95,
 port_no=2, LIVE> Port<dpid=95, port_no=3, LIVE> >
SwitchCounter: active switches = 95
```

(a) Step C. The controller log shows that the reconnections are captured, but no state transition is performed.



```
DisasterMonitorApp: Switch<dpid=100, Port<dpid=100, port_no=1, LIVE> Port<dpid=1
00, port_no=2, LIVE> Port<dpid=100, port_no=3, LIVE> >
SwitchCounter: active switches = 100
IdManagerApp: Received event: EventSwitchEnter
IdManagerApp: Table miss flow entry set
IdManagerApp: Has ID flow entries set
IdManagerApp: Attach ECS ID flow entries set. Total IPs: 1
IdManagerApp: No ID flow entry set
DisasterMonitorApp: Received event: EventSwitchEnter
DisasterMonitorApp: Switch<dpid=101, Port<dpid=101, port_no=1, LIVE> Port<dpid=1
01, port_no=2, LIVE> Port<dpid=101, port_no=3, LIVE> >
SwitchCounter: active switches = 101
IdManagerApp: Received event: EventSwitchEnter
IdManagerApp: Table miss flow entry set
IdManagerApp: Has ID flow entries set
IdManagerApp: Attach ECS ID flow entries set. Total IPs: 1
IdManagerApp: No ID flow entry set
DisasterMonitorApp: Received event: EventSwitchEnter
DisasterMonitorApp: Switch<dpid=102, Port<dpid=102, port_no=1, LIVE> Port<dpid=1
02, port_no=2, LIVE> Port<dpid=102, port_no=3, LIVE> >
SwitchCounter: active switches = 102
NormalOperationState: created
DisasterMonitorApp: Applied new state
IdManagerApp: Received event: EventSwitchEnter
```

(b) Step D. The controller log shows that the *exit disaster* threshold is reached and the state transition is performed.

Figure 6.11: Controller log generated when processing steps C and D of the multiple switch recovery test.

6.4.2.2 Traffic congestion

To simulate a traffic congestion scenario, we need a network topology and data streams that will saturate one or more links. A minimalistic setup that shows the scenario is shown in Figure 6.12, where the link that connects switches *s1* and *s2* is the focal point of study. For this setup, all links were configured to support a bandwidth of 100 Mbit/s and the hosts have IPs addresses that represent ECS, Disaster and Non-Disaster located users.

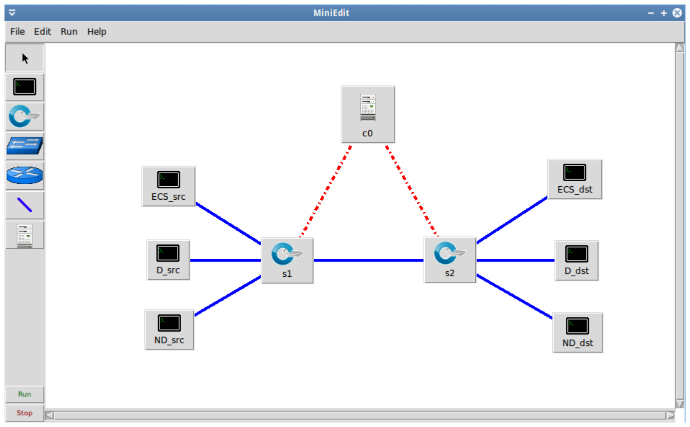


Figure 6.12: Minimalistic topology used to simulate traffic congestion, visualized through Miniedit, a GUI editor for Mininet.

For easier setup, the author implemented the script *traffic-minimal.sh* which firstly cleans up Mininet and Open VSwitch to then call *traffic-test-minimal.py*, which is a Python script that performs the following steps:

1. Builds and configure the Mininet topology.
2. Uses *set_bridges.sh* script to configure Open VSwitch.
3. Uses *minimal_ryu.sh* to launch the Ryu controller with the designed applications.
4. Performs a ping test.
5. Starts the iPerf servers which acts as data receivers.
6. Starts the iPerf clients which send data to the servers.

The controller's configuration and the sender's data stream bandwidth were modified to create different scenarios. As Mininet is a non-deterministic simulator, each experiment was performed five times to create reliable and stabilized data.

The graphical representations depicted in Figures 6.13, 6.14 and 6.15, present a summary of the results obtained from the experiments which will be described in this Section. The primary objective of this visual summary is to facilitate a straightforward comparison of the findings in a user-friendly format. The comprehensive results of each experiment are presented in a graphical form in Annex B.

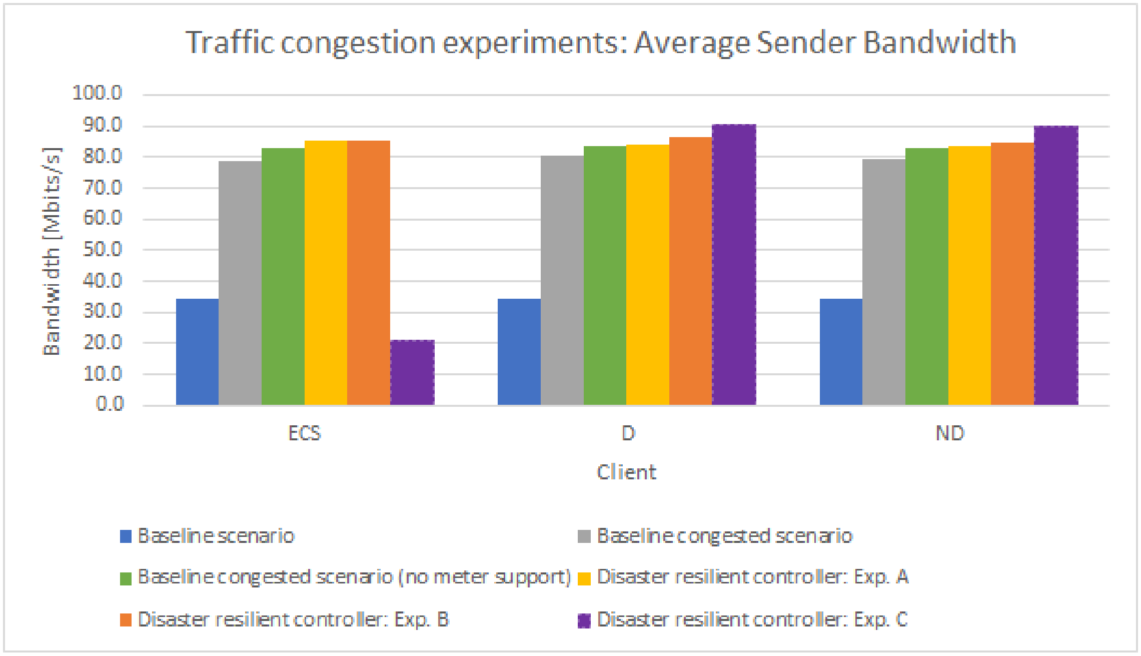


Figure 6.13: Summary of the results obtained from the traffic control experiments which shows the average bandwidth measured from the sender in Mbits/s. Communication types are ECS, disaster (D) and non-disaster (ND).

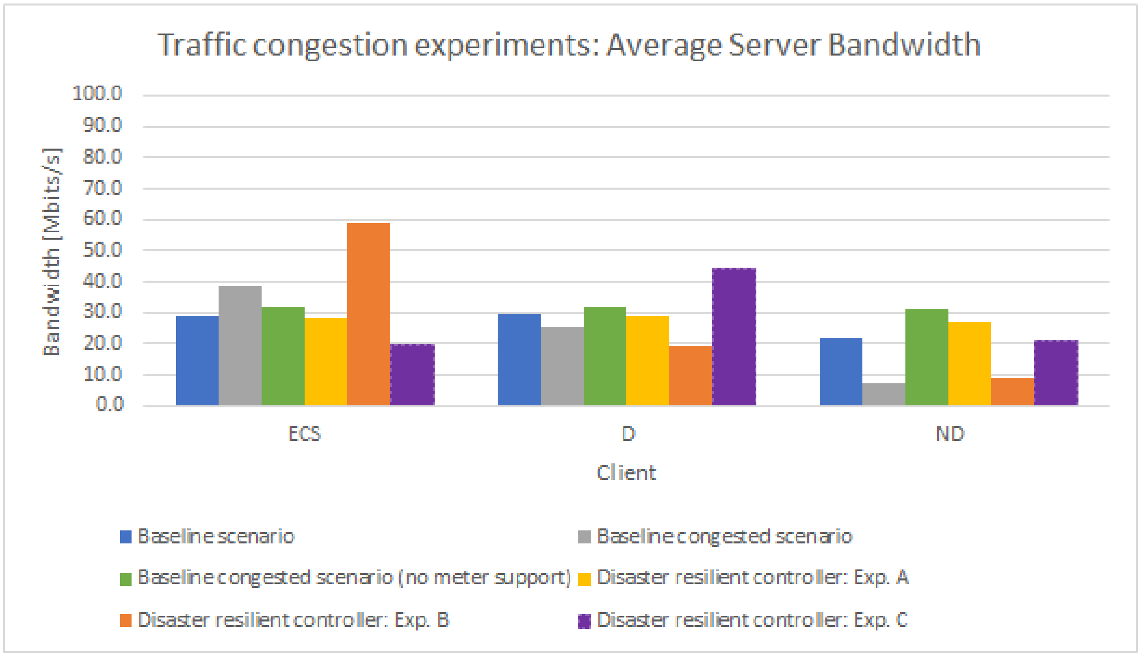


Figure 6.14: Summary of the results obtained from the traffic control experiments which shows the average bandwidth measured in the receiver side in Mbits/s. Communication types are ECS, disaster (D) and non-disaster (ND).

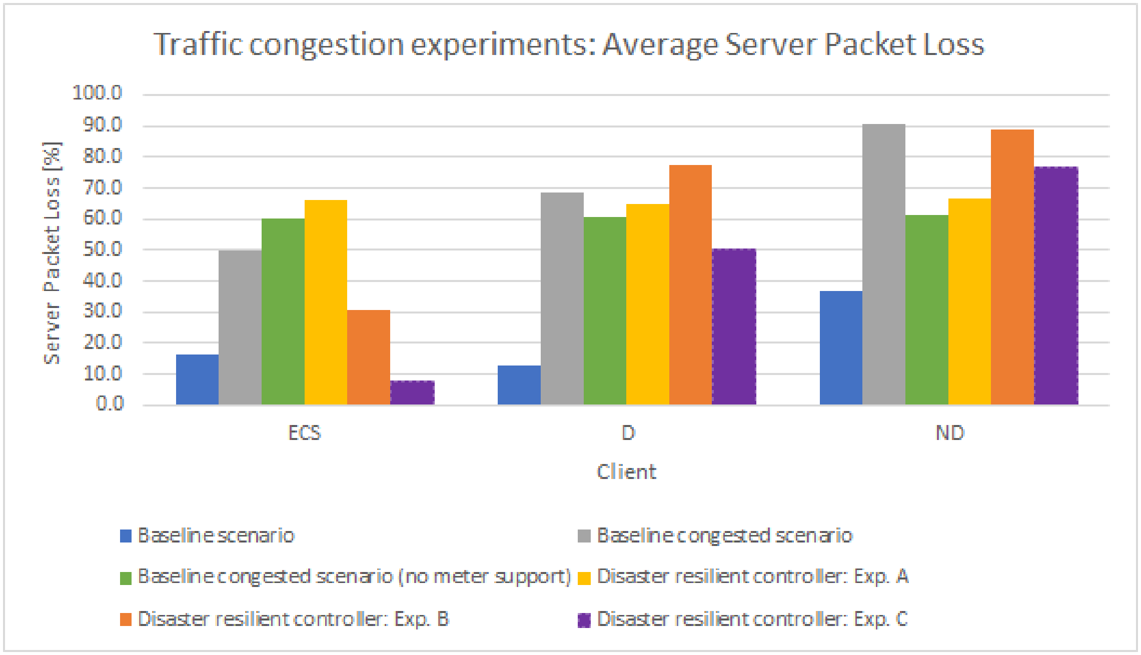


Figure 6.15: Summary of the results obtained from the traffic control experiments which shows the average packet loss percentage of the communications. Communication types are ECS, disaster (D) and non-disaster (ND).

6.4.2.2.1 Baseline scenario

For the baseline scenario we use a *default controller* without disaster configuration, i.e. the controller does not apply any QoS feature and only performs routing management. The senders each create data streams of 33 Mbits/s, meaning the *s1-s2* link would not be congested. Figure B.1 (a) and (b) show the measured bandwidth on each end user of the communications.

On the sender side, all senders' average bandwidth ranges from 34.1 to 34.2 Mbits/s and the median ranges from 34.1 to 34.3 Mbits/s. These values are 1 Mbit/s higher than the configured bandwidth value for the data stream, but close enough to consider them accurate.

On the receiver side, the expectation was to measure a bandwidth of 33 Mbits/s across all receivers given that the *s1-s2* link should not become a bottleneck. However, we see that the measured bandwidth for the non-disaster communication is 21.7 Mbits/s on average and has a median of 20.6 Mbits/s, lower than expected.

Figure B.1 (c) shows the packet loss percentage on each communication. Non-disaster communications showing a higher packet loss percentage are consistent with the lower bandwidth measured.

The author suspects that a default traffic control configuration on the switches might be causing the unexpected results. The author proceeded with the next experiments to determine if the measured bandwidth on any of the receivers continued to exhibit lower values.

6.4.2.2.2 Baseline congested scenario

For the baseline congested scenario we again use a *default controller* but now each sender creates data streams of 100 Mbits/s, creating traffic congestion on the *s1-s2* link. Figure B.2 (a) and (b) shows the measured bandwidth on each end user of the communications.

On the sender side, all senders' average bandwidth ranges from 78.8 to 80.4 Mbits/s and the median ranges from 76.5 to 80.5 Mbits/s. While lower than the configured bandwidth to be sent, there is no major difference between senders.

On the receiver side, the measured results vary across each communication type. ECS communications show the higher bandwidth averaging 38.8 Mbits/s, disaster communications show an average bandwidth of 25.3 Mbits/s and non-disaster communications show an average bandwidth of 7.3 Mbits/s. These results show a non-equal bandwidth allocation.

The *s1-s2* link is successfully acting as the bottleneck, thus effectively limiting the total aggregated bandwidth of all communications to a value lower than 100 Mbits/s.

The non-equal bandwidth distribution shows again a lower allocation for non-disaster communications, favoring ECS communications the most. This result is consistent through all runs. The author researched this issue and found a limitation between Mininet and Open VSwitch setup. The following comment found in Mininet's source code explains the situation: *"Unfortunately OVS and Mininet are fighting over tc queuing disciplines. As a quick hack/workaround, we clear OVS's and reapply our own."*.

The script *set_ bridges.sh* is used to configure the switches with Open VSwitch commands after Mininet has finished its configuration step, rendering the hack ineffective. This Open VSwitch configuration is needed for meter support.

To verify if the conflict in traffic control configurations between Mininet and Open VSwitch was indeed the issue, a similar test was conducted with the exception of skipping the configurations that enable support for meters (i.e., not using *set_ bridges.sh*). The results of this experiment are depicted in Figure B.3. The receiver's measured bandwidth for ECS, disaster and non-disaster communications are 31.9 Mbits/s, 32.0 Mbits/s and 31.3 Mbits/s on average, respectively; and the median is 31.9 Mbits/s, 32.6 Mbits/s and 30.1 Mbits/s respectively. It can be observed that the bandwidth is evenly distributed among all communications, thus confirming that the conflict between Mininet and Open vSwitch configurations impacts the traffic control management policies of the switches.

The next question is whether the QoS policies implemented by the proposed disaster resilient controller in this study are sufficient to circumvent the conflict between Mininet and Open vSwitch, and if the data generated from the experiments conducted on this testbed can be deemed reliable. Experiment A in the following Section will specifically address this question.

6.4.2.2.3 Disaster resilient controller in congested scenarios

For this scenario the controller will apply its disaster QoS policies defined by the proposed applications. Different experiments were tested in this scenario by modifying the bandwidth of the data streams created by the senders and also by setting up different rate limiting policies on the switches using the Prioritization application. The values used are summarized in Table 6.3.

Experiment	Data stream per communication type [Mbits/s]			Controller's rate limit per communication type [Mbits/s]		
	ECS	Disaster	Non-Disaster	ECS	Disaster	Non-Disaster
A	100	100	100	33	33	33
B	100	100	100	70	20	10
C	20	100	100	20	50	30

Table 6.3: Data stream bandwidths and rate limit policies used to test the designed disaster resilient controller. The rate limits are enforced by the Prioritization application.

Experiment A is designed to assign an equal bandwidth distribution between all communication types, with the purpose of determining if the meter management performed by the controller is enough to override the conflict of traffic control configurations between Mininet and Open VSwitch. With each sender creating data streams of 100 Mbits/s and having a rate limiting policy of 33 Mbits/s on all communication types set by the controller, we can directly compare the results with the Baseline congested scenario shown in the previous Section. Figure B.4 shows the measured results for experiment A.

The results for experiment A show that the receivers' measured bandwidth for ECS, disaster and non-disaster communications are 28.62 Mbits/s, 29.22 Mbits/s and 27.04 Mbits/s on average, respectively; and the median is 28.8 Mbits/s, 28.8 Mbits/s and 27.2 Mbits/s respectively.

We see that the controller rate limiting policies from the Prioritization application took effect, meaning the OpenFlow commands were able to configure the switches through the Open VSwitch API, solving the configuration conflicts with Mininet that prevented achieving a fair distribution of the bandwidth.

It is noteworthy that in the Baseline congested scenario where Mininet solely manages the traffic control policies on the switches, the aggregated measured bandwidth on the receivers is closer to the limit of 100Mbits/s set by the *s1-s2* link, compared to the aggregated total bandwidth measured on the receivers of experiment A. The author concludes from this result that solely using OpenFlow meters for QoS does not allow to achieve an optimal bandwidth usage, and further investigating other OpenFlow features for QoS management, such as OpenFlow switches' queue management, would be worthwhile. This is proposed as future work in Section 7.2.

Experiment B is designed to highly favor one communication type, in particular ECS. The measured results are shown in Figure B.5. The results show that the receivers measured bandwidth for ECS, disaster and non-disaster communications are 59.0 Mbits/s, 19.2 Mbits/s and 9.2 Mbits/s on average, respectively; and the median is 59.0 Mbits/s, 19.2 Mbits/s and 9.3 Mbits/s respectively. Packet loss for ECS, disaster and non-disaster communications shows an average of 30.6%, 77.4% and 88.8% respectively.

We again see a non-optimal bandwitdh usage with the aggregated total bandwidth measured on the receivers being lower than the limit enforced by the *s1-s2* link. In particular, ECS communications measured bandwidth on the receiver is 11 Mbits/s lower than the applied rate limit of 70 Mbits/s, meaning it only made use of 84% of its reserved bandwidth. Disaster and non-disaster communications use 96% and 93% of their allowance, respectively.

The author concludes from this experiment that the differentiated bandwidth rate limit policies were correctly applied by the controller. If we compare experiment A and B, we see that the rate limit configuration enforced by the controller in experiment B effectively improves QoS for ECS communications, reducing the packet loss from 66.2% to 30.6%.

Experiment C is designed to simulate a more realistic scenario, where ECS communications present with a lower rate than disaster or non-disaster communications. In this case, the rate limiting policy from the controller tries to allocate full bandwidth to ECS communications, to then favor disaster communications over non-disaster communications. The simulation results are shown in Figure B.6.

The results for experiment C show that the receivers' measured bandwidth for ECS, disaster and non-disaster communications are 19.4 Mbits/s, 44.4 Mbits/s and 21.0 Mbits/s on average, with a median of 19.4 Mbits/s, 44.1 Mbits/s and 21.1 Mbits/s.

We see that the reserved bandwidth policies for ECS communications were effectively applied, achieving 97% usage and only having an average packet loss of 7.6%. Out of the remaining bandwidth, disaster communications were correctly favored by the controller.

6.5 Limitations

While the testing of the disaster resilient controller over the Mininet and OVS testbed has yielded positive results, some limitations were identified.

One limitation observed was the conflict between Mininet and Open VSwitch over traffic control disciplines on the switches. Care must be taken to avoid such conflicts, and further investigation may be needed to ensure smooth functioning of traffic control policies in the testbed.

Another limitation found by the author relates to the creation of a network that uses more than one controller. While Mininet easily allows the configuration of the communication channel between the switches and the controllers, the author was unable to configure a communication channel between the controllers themselves. This limitation has hindered the testing of the Disaster Ex Situ state and its notification system.

Additionally, the routing application provided by Ryu was found to be simplistic and lacking dynamic rerouting capabilities, which prevented the testing of bandwidth measurements in a constantly changing network. As a future improvement, implementing a dynamic routing application for the controller could address this limitation.

6.6 Conclusions

The testing of the three designed applications in the testbed using Mininet and OVS has shown promising results. The functionalities of the applications were tested in both simple and complex scenarios, and the designed applications were found to work effectively with other applications (in this particular case, with the routing application).

The designed disaster resilient controller demonstrated its ability to detect disasters, manage different levels of prioritization, and apply disaster QoS policies dynamically as needed. The rate limiting policies were found to be effective in controlling the usage of bandwidth. However, further investigation into QoS queuing policies may be needed to optimize the usage of assigned bandwidth.

The testbed using Mininet and Open VSwitch proved to be a capable tool for simulating SDNs, despite some limitations. While Mininet is not a deterministic simulator, the consistency of results between runs provides reliability. The conflicts between Mininet and Open VSwitch over traffic control policies were overcome by the QoS management performed by the controller. However, the inability to configure communication between two controllers in the testbed is a limitation that could be addressed in future work.

The identified limitations require further attention and improvement for more comprehensive and realistic testing in the future. Addressing these limitations will contribute to evaluating the robustness and reliability of the designed applications of the disaster resilient controller, ensuring their effective performance in managing disasters and applying QoS policies.

Overall, the testing of the disaster resilient controller proposed in this work has provided valuable insights into the SDN functionalities, performance, and limitations of the existing tools.

Chapter 7

Conclusions

7.1 Main contributions

The systematic review on software-defined networks (SDNs) in the context of disaster resilient WANs has provided a comprehensive overview of the state of the art in SDNs, their existing applications in wide area networks (WANs), and the benefits and challenges associated with using them to create disaster resilient networks. The review also identified useful tools, APIs, and frameworks for open source SDN development.

With the information gathered, we were able to successfully answer the research questions that motivated this study. Firstly, we found that SDNs can be applied to WANs as long as the original concept of SDNs is enhanced to solve the scalability, performance, reliability, and security issues found. We also found that SDN applicability over WANs has been widely studied, and multiple solutions to these issues have been proposed. Secondly, we identified that the control plane design is key to achieving acceptable levels of QoS and resilience. This encompasses physical components such as the controller's placement and its capacity to handle requests, as well as software components such as controller applications that handle management, routing, and recovery algorithms. Lastly, we reviewed how disaster scenarios affect traditional WANs and the specific challenges they present for normal operation. In addition, we found SDN-WAN studies aiming to tackle and handle those challenges.

As a result of this review, we found that the methodologies proposed in the reviewed articles work on isolation and combining techniques is not explored. We also found unanswered questions regarding when and how disaster configuration should be triggered in an SDN-WAN controller and how this controller can automatically switch between normal and disaster configurations. Furthermore, we found that implementation details of the controller applications were not explained.

With this unexplored field identified, this study aimed to propose answers to these open questions by designing, implementing, and testing an SDN controller with a specific focus on disaster resilience.

The SDN architecture has demonstrated its effectiveness in providing features and tools to combat disasters, as well as implementing new techniques that may not be easily applicable in traditional networks. The extensibility of the designed SDN controller allows for easy addition of other applications and features to the controller, making it adaptable to evolving needs.

The identification of simulation techniques for testing the SDN controller, as well as the simulation of disaster scenarios, has provided valuable insights into the controller's performance and resilience in various scenarios. However, it is important to acknowledge the limitations of the testbed used for simulation, which should be considered when interpreting the results.

The validation of the designed controller's features has shown that the dynamic monitoring and QoS management features are valuable in designing a disaster resilient controller for WANs. Therefore, this research has made a significant contribution to the body of knowledge on SDNs and their potential applications in disaster-resilient networks.

7.2 Future and proposed work

Further research and refinement of the designed SDN controller for disaster resilience can greatly contribute to the development of more robust and efficient disaster resilient networks. There are several areas that could benefit from additional investigation and optimization.

Firstly, optimal parameters and thresholds for state change triggers should be identified to improve the controller's ability to detect and respond to disaster events. This could involve determining the most appropriate values for parameters such as threshold levels for state change triggers, both for initiating a disaster state from normal operation and vice versa. Fine-tuning these parameters can enhance the accuracy and timeliness of the controller's response to disaster events.

Additionally, considering user input as a possible trigger for state changes could enhance the usability and practicality of the controller. User input, such as manual override or input from emergency responders, can provide valuable insights and enable more informed decision-making during disaster events. Incorporating user input into the controller's operation can enhance its responsiveness and adaptability in real-world disaster scenarios.

Secondly, meter bands and other parameters used for dynamic adjustment within the controller could be further optimized. This could involve investigating different meter band configurations, adjusting parameters based on network conditions, and dynamically adapting the controller's actions accordingly. This can improve the efficiency and effectiveness of the controller in managing network resources during disaster events.

In addition, leveraging OpenFlow switches' queue management capabilities to enhance QoS effectiveness should be further researched. Investigating different queue management strategies, optimizing queue configurations, and integrating QoS management into the controller's disaster resilience features can improve the network's ability to prioritize critical traffic during disaster events, ensuring efficient resource utilization and maintaining service quality for final users.

Thirdly, testing setups that enable communication between two or more controllers should be explored. This would enable the testing of the proposed notification system between remote controllers, enabling the Disaster Ex Situ state feature and allowing for coordinated actions and responses in case of large-scale disasters that span across multiple network domains. Such communication capabilities can enhance the scalability and effectiveness of the controller in managing disaster events.

Moreover, testing the controller with dynamic routing applications can provide insights into its compatibility and effectiveness in real-world networking scenarios. Dynamic routing apps can introduce additional complexities and challenges in managing network resources during disaster events, and testing the controller in such scenarios can help refine its performance and resilience.

In conclusion, further research and refinement of the SDN controller's features, performance and testing can lead to more robust and efficient disaster resilient networks in the future. Investigating optimal parameters, thresholds, user input integration, testing in real-scale networks, communication between controllers, and leveraging OpenFlow switches' capabilities can contribute to enhancing the controller's effectiveness in managing disaster events and improving the overall resilience of SDN-based networks.

Bibliography

[1] Ryu application API, 2017. `https://ryu.readthedocs.io/en/latest/ryu_app_api.html` Accessed: 2023-03-15.

[2] NOX github repository, 2020. `https://github.com/noxrepo/pox` Accessed: 2023-03-15.

[3] Ameer Mosa Al-Sadi, Ali Al-Sherbaz, James Xue, and Scott Turner. Routing algorithm optimization for software defined network wan. In *2016 Al-Sadeq International Conference on Multidisciplinary in IT and Communication Science and Applications (AIC-MITCSA)*, pages 1–6, 2016.

[4] Suleiman Onimisi Aliyu, Feng Chen, Ying He, and Hongji Yang. A game-theoretic based qos-aware capacity management for real-time edgeiot applications. In *2017 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pages 386–397, 2017.

[5] Saeed A. Astaneh and Shahram Shah Heydari. Optimization of sdn flow operations in multi-failure restoration scenarios. *IEEE Transactions on Network and Service Management*, 13(3):421–432, 2016.

[6] Cristian Hernandez Benet, Kyoomars Alizadeh Noghani, and Andreas J. Kassler. Minimizing live vm migration downtime using openflow based resiliency mechanisms. In *2016 5th IEEE International Conference on Cloud Networking (Cloudnet)*, pages 27–32, 2016.

[7] Marcin Bienkowski, Anja Feldmann, Johannes Grassler, Gregor Schaffrath, and Stefan Schmid. The wide-area virtual service migration problem: A competitive analysis approach. *IEEE/ACM Transactions on Networking*, 22(1):165–178, 2014.

[8] Franco Callegati, Walter Cerroni, Chiara Contoli, and Francesco Foresta. Performance of intent-based virtualized network infrastructure management. In *2017 IEEE International Conference on Communications (ICC)*, pages 1–6, 2017.

[9] Giuseppe Carella, Junnosuke Yamada, Niklas Blum, Christian Lück, Naoyoshi Kanamaru, Naoki Uchida, and Thomas Magedanz. Cross-layer service to network orchestration. In *2015 IEEE International Conference on Communications (ICC)*, pages 6829–6835, 2015.

[10] Xiangqing Chang, Jun Li, Guodong Wang, Zexin Zhang, Lingling Li, and Yalin Niu. Software defined backpressure mechanism for edge router. In *2015 IEEE 23rd International Symposium on Quality of Service (IWQoS)*, pages 171–176, 2015.

[11] Ryu SDN Framework Community. Ryu project website, 2017. `https://ryu-sdn.org/` Accessed: 2023-03-15.

[12] Boeing Company. Core documentation, 2022. `http://coreemu.github.io/core/` Accessed: 2023-03-15.

[13] Mininet Project Contributors. Mininet project website, 2022. `http://mininet.org/overview/` Accessed: 2023-03-15.

[14] Raheleh Dilmaghani and Dae Kwon. Evaluation of openflow load balancing for navy. In *MILCOM 2015 - 2015 IEEE Military Communications Conference*, pages 133–138, 2015.

[15] George Elmasry, Diane McClatchy, Rick Heinrich, and Kevin Delaney. A software defined networking framework for future airborne connectivity. In *2017 Integrated Communications, Navigation and Surveillance Conference (ICNS)*, pages 2C2–1–2C2–9, 2017.

[16] Ilhem Fajjari, Nadjib Aitsaadi, and Djamel Eddine Kouicem. A novel sdn scheme for qos path allocation in wide area networks. In *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, pages 1–7, 2017.

[17] The Linux Foundation. netem, 2023. `https://wiki.linuxfoundation.org/networking/netem` Accessed: 2023-03-15.

[18] O. N. Fundation. Openflow switch specification: Version 1.3.0., 2012. `https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf` Accessed: 2023-03-15.

[19] O. N. Fundation. Openflow switch specification: Version 1.5.1., 2015. `https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf` Accessed: 2023-03-15.

[20] Pablo L. Gallegos-Segovia, Jack F. Bravo-Torres, Víctor M. Larios-Rosillo, Paúl E. Vintimilla-Tapia, Iván F. Yuquilima-Albarado, and Santiago J. Arévalo-Cordero. Living lab concept for cloud analysis in networks of metropolitan sensors applying the concept of sd-wan and hybrid networks. In *2017 CHILEAN Conference on Electrical, Electronics Engineering, Information and Communication Technologies (CHILECON)*, pages 1–6, 2017.

[21] Pablo L. Gallegos-Segovia, Jack F. Bravo-Torres, Paúl E. Vintimilla-Tapia, Jorge O. Ordoñez-Ordoñez, Ruben E. Mora-Huiracocha, and Victor M. Larios-Rosillo. Evaluation of an sdn-wan controller applied to services hosted in the cloud. In *2017 IEEE Second Ecuador Technical Chapters Meeting (ETCM)*, pages 1–6, 2017.

[22] Yu Hua, Wenbo He, Xue Liu, and Dan Feng. Smarteye: Real-time and efficient cloud

image sharing for disaster environments. In *2015 IEEE Conference on Computer Communications (INFOCOM)*, pages 1616–1624, 2015.

[23] Melih A. Karaman, Burak Gorkemli, Sinan Tatlicioglu, Mustafa Komurcuoglu, and Ozgur Karakaya. Quality of service control and resource prioritization with software defined networking. In *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*, pages 1–6, 2015.

[24] Wisarut Khumngoen, Wanida Putthividhya, and Vasuwat Tan-anannuwat. Fine-grained bandwidth allocation in software-defined networks. In *2016 International Computer Science and Engineering Conference (ICSEC)*, pages 1–6, 2016.

[25] Barbara Kitchenham. Procedures for performing systematic reviews. *Keele, UK, Keele University*, 33(2004):1–26, 2004.

[26] Martin Klapez, Carlo Augusto Grazia, and Maurizio Casoni. Towards massively multipath transmissions for public safety communications. In *2016 IEEE 12th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pages 1–7, 2016.

[27] Stanislav Lange, Steffen Gebert, Joachim Spoerhase, Piotr Rygielski, Thomas Zinner, Samuel Kounev, and Phuoc Tran-Gia. Specialized heuristics for the controller placement problem in large scale sdn networks. In *2015 27th International Teletraffic Congress*, pages 210–218, 2015.

[28] Sabita Maharjan, Yan Zhang, Stein Gjessing, Oystein Ulleberg, and Frank Eliassen. Providing microgrid resilience during emergencies using distributed energy resources. In *2015 IEEE Globecom Workshops (GC Wkshps)*, pages 1–6, 2015.

[29] Milos Manic, Dumidu Wijayasekara, Kasun Amarasinghe, Joel Hewlett, Kevin Handy, Christopher Becker, Bruce Patterson, and Robert Peterson. Next generation emergency communication systems via software defined networks. In *2014 Third GENI Research and Educational Experiment Workshop*, pages 1–8, 2014.

[30] Hellen Maziku and Sachin Shetty. Software defined networking enabled resilience for iec 61850-based substation communication systems. In *2017 International Conference on Computing, Networking and Communications (ICNC)*, pages 690–694, 2017.

[31] F. Mendoza, R. Ferrús, and O. Sallent. Sdn-based traffic engineering for improved resilience in integrated satellite-terrestrial backhaul networks. In *2017 4th International Conference on Information and Communication Technologies for Disaster Management (ICT-DM)*, pages 1–8, 2017.

[32] Red Universtaria Nacional. Reuna digital infrastructure 2018, 2023. `https://www.reuna.cl/infraestructura-digital/#red-nacional-2018` Accessed: 2023-03-15.

[33] Hyunwoo Nam, Doru Calin, and Henning Schulzrinne. Intelligent content delivery over wireless via sdn. In *2015 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 2185–2190, 2015.

[34] H Sofia Naning, Rendy Munadi, and Muhammad Zen Effendy. Sdn controller placement design: For large scale production network. In *2016 IEEE Asia Pacific Conference on Wireless and Mobile (APWiMob)*, pages 74–79, 2016.

[35] Bram Naudts, Marlies Van der Wee, Luc Andries, Sofie Verbrugge, and Didier Colle. Techno-economic analysis of a software-defined optical media contribution and distribution network. In *2016 8th International Workshop on Resilient Networks Design and Modeling (RNDM)*, pages 116–121, 2016.

[36] Kien Nguyen, Quang Tran Minh, and Shigeki Yamada. Increasing resilience of openflow wans using multipath communication. In *2013 International Conference on IT Convergence and Security (ICITCS)*, pages 1–2, 2013.

[37] Kien Nguyen, Quang Tran Minh, and Shigeki Yamada. A software-defined networking approach for disaster-resilient wans. In *2013 22nd International Conference on Computer Communication and Networks (ICCCN)*, pages 1–5, 2013.

[38] Kien Nguyen, Quang Tran Minh, and Shigeki Yamada. Towards optimal disaster recovery in backbone networks. In *2013 IEEE 37th Annual Computer Software and Applications Conference*, pages 826–827, 2013.

[39] Van-Quyet Nguyen, Sinh Ngoc Nguyen, and Kyungbaek Kim. Enabling disaster-resilient sdn with location trustiness. In *2017 4th International Conference on Computer Applications and Information Processing Technology (CAIPT)*, pages 1–4, 2017.

[40] nsnam. ns-3 network simulator website, 2023. `https://www.nsnam.org/` Accessed: 2023-03-15.

[41] Mathis Obadia, Mathieu Bouet, Jérémie Leguay, Kévin Phemius, and Luigi Iannone. Failover mechanisms for distributed sdn controllers. In *2014 International Conference and Workshop on the Network of the Future (NOF)*, volume Workshop, pages 1–6, 2014.

[42] Koichi Ogawa and Noriaki Yoshiura. Network operational method by using software-defined networking for improvement of communication quality at disasters. In *The 16th Asia-Pacific Network Operations and Management Symposium*, pages 1–4, 2014.

[43] Camila Faundez Orellana. Disaster resilient SDN controller github repository, 2023. `https://github.com/cfaundez/DisasterResilientSdnController` Accessed: 2023-04-18.

[44] Nancy Perrot and Thomas Reynaud. Optimal placement of controllers in a resilient sdn architecture. In *2016 12th International Conference on the Design of Reliable Communication Networks (DRCN)*, pages 145–151, 2016.

[45] K. Phemius and M. Bouet. Implementing openflow-based resilient network services. In *2012 IEEE 1st International Conference on Cloud Networking (CLOUDNET)*, 2012.

[46] Kévin Phemius, Mathieu Bouet, and Jérémie Leguay. Disco: Distributed multi-domain sdn controllers. In *2014 IEEE Network Operations and Management Symposium*

(NOMS), pages 1–4, 2014.

[47] Kévin Phemius, Mathieu Bouet, and Jérémie Leguay. Disco: Distributed sdn controllers in a multi-domain environment. In *2014 IEEE Network Operations and Management Symposium (NOMS)*, pages 1–2, 2014.

[48] V Ramiro, J Piquer, T Barros, and P Sepúlveda. The chilean internet: Did it survive the earthquake? *WIT Transactions on State-of-the-art in Science and Engineering*, 58, 2012.

[49] Ali Sanhaji, Philippe Niger, Philippe Cadro, Cédric Ollivier, and André-Luc Beylot. Congestion-based api for cloud and wan resource optimization. In *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, pages 141–145, 2016.

[50] Goshi Sato, Noriki Uchida, and Yoshitaka Shibata. Performance evaluation of software defined and cognitive wireless network based disaster resilient system. In *2015 IEEE 29th International Conference on Advanced Information Networking and Applications*, pages 741–746, 2015.

[51] Goshi Sato, Noriki Uchida, Norio Shiratori, and Yoshitaka Shibata. Performance analysis of never die network based on software defined disaster resilient system. In *2015 18th International Conference on Network-Based Information Systems*, pages 64–70, 2015.

[52] S. Sedef Savas, Massimo Tornatore, M. Farhan Habib, Pulak Chowdhury, and Biswanath Mukherjee. Disaster-resilient control plane design and mapping in software-defined networks. In *2015 IEEE 16th International Conference on High Performance Switching and Routing (HPSR)*, pages 1–6, 2015.

[53] Abhishek Singh, Mayank Tiwray, Raj Kumar, and Rachita Misra. Load balancing among wide-area sdn controllers. In *2016 International Conference on Information Technology (ICIT)*, pages 104–109, 2016.

[54] Maryam Tanha, Dawood Sajjadi, and Jianping Pan. Enduring node failures through resilient controller placement for software defined networks. In *2016 IEEE Global Communications Conference (GLOBECOM)*, pages 1–7, 2016.

[55] Edion Tego, Francesco Matera, Vincenzo Attanasio, and Donato Del Buono. Quality of service management based on software defined networking approach in wide gbe networks. In *2014 Euro Med Telco Conference (EMTC)*, pages 1–5, 2014.

[56] Markku Vajaranta, Joona Kannisto, and Jarmo Harju. Implementation experiences and design challenges for resilient sdn based secure wan overlays. In *2016 11th Asia Joint Conference on Information Security (AsiaJCIS)*, pages 17–23, 2016.

[57] L. Vdovin, P. Likin, and A. Vilchinskii. Network utilization optimizer for sd-wan. In *2014 International Science and Technology Conference (Modern Networking Technologies) (MoNeTeC)*, pages 1–4, 2014.

[58] An Xie, Xiaoliang Wang, Wei Wang, and Sanglu Lu. Designing a disaster-resilient

network with software defined networking. In *2014 IEEE 22nd International Symposium of Quality of Service (IWQoS)*, pages 135–140, 2014.

[59] Huan Yan, Jiaqiang Liu, Yong Li, Wenxia Dong, Chengyong Lin, and Depeng Jin. Wan as a service for cloud via software-defined network and open apis. In *2015 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 9–10, 2015.

[60] Haruka Yanagida, Akihiro Nakao, Shu Yamamoto, Saneyasu Yamaguchi, and Masato Oguchi. Sns information-based network control system developed on flare experiment environment. In *2016 IEEE 17th International Conference on High Performance Switching and Routing (HPSR)*, pages 120–121, 2016.

Annexes

Annex A

Systematic review: Topic Summary

Cite	Topic
[46]	Distributed SDN Control plane for WAN
[5]	SD-WAN for disaster scenario
[44]	SDN Controller placement
[22]	SD-WAN application
[54]	SDN Controller placement
[47]	Distributed SDN Control plane for WAN
[27]	SDN Controller placement
[41]	SD-WAN resilience
[7]	SD-WAN application
[52]	SDN Controller placement
[37]	SD-WAN for disaster scenario
[58]	SD-WAN for disaster scenario
[30]	SD-WAN application
[8]	SD-WAN application
[3]	SD-WAN routing
[29]	SD-WAN for disaster scenario
[26]	SD-WAN for disaster scenario
[6]	SD-WAN for disaster scenario
[57]	SD-WAN routing
[23]	SD-WAN for disaster scenario
[16]	SD-WAN QoS
[55]	SD-WAN QoS
[9]	SDN QoS
[21]	SD-WAN application

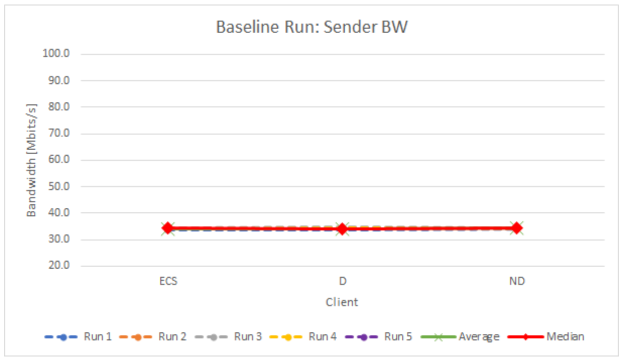
Cite	Topic
[53]	SD-WAN resource optimization
[50]	SD-WAN for disaster scenario
[10]	SD-WAN resilience
[4]	SDN QoS
[49]	SD-WAN resource optimization
[14]	SD-WAN resource optimization
[33]	SDN QoS
[20]	SD-WAN application
[34]	SDN Controller placement
[31]	SD-WAN resilience
[56]	SD-WAN Security, SD-WAN resilience
[36]	SD-WAN resilience
[51]	SD-WAN resilience
[38]	SD-WAN resilience
[42]	SD-WAN for disaster scenario
[35]	SD-WAN QoS
[15]	SD-WAN Security
[39]	SD-WAN for disaster scenario
[24]	SD-WAN resource optimization
[60]	SD-WAN for disaster scenario
[59]	SD-WAN application

Table A.1: Topic summary of the approved studies for the Systematic Literature review.

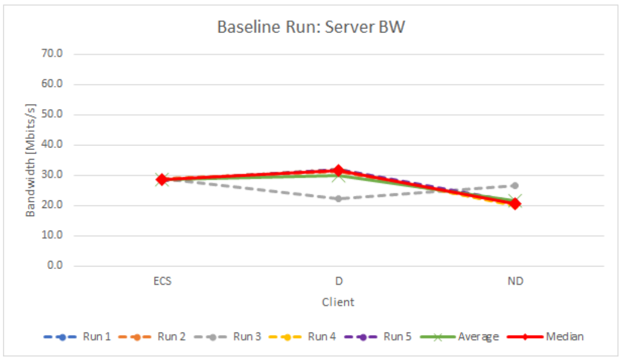
Annex B

Results of traffic congestion experiments

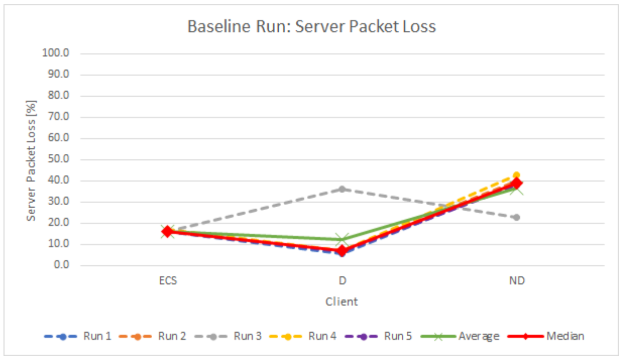
B.1 Baseline scenario



(a) Measured bandwidth on sender's end.



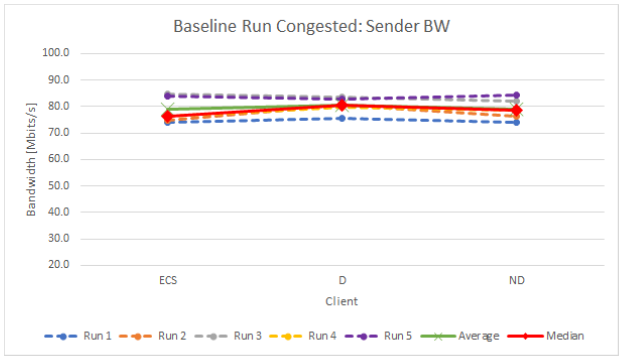
(b) Measured bandwidth on receiver's end.



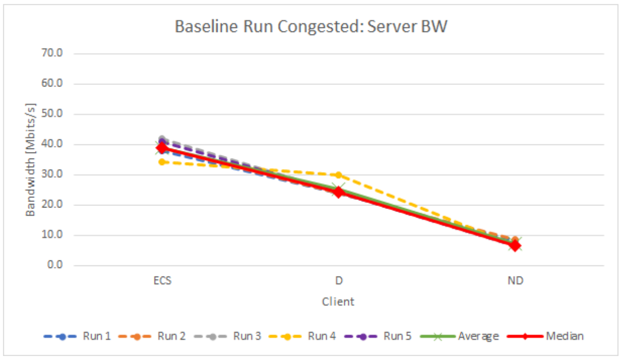
(c) Measured packet loss on receiver's end.

Figure B.1: Baseline scenario. Subfigures (a) and (b) show the measured bandwidth on each end user of the communication, and subfigure (c) shows the packet loss of each communication type. iPerf is configured to create UDP streams of 33 Mbits/s on each sender. The experiment was performed five times and the graphs show the measured values of each run, the average value and the median.

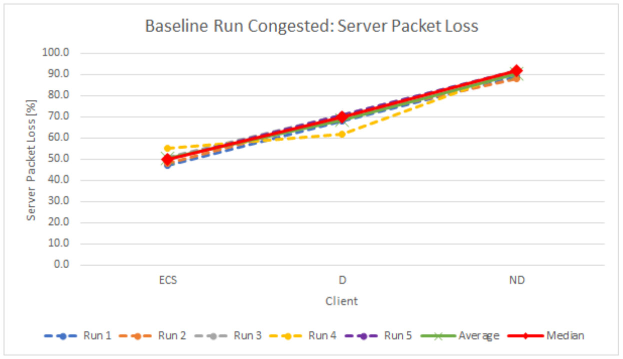
B.2 Baseline congested scenario



(a) Measured bandwidth on sender's end.



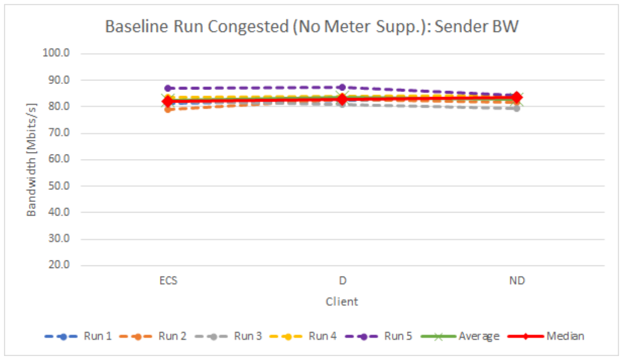
(b) Measured bandwidth on receiver's end.



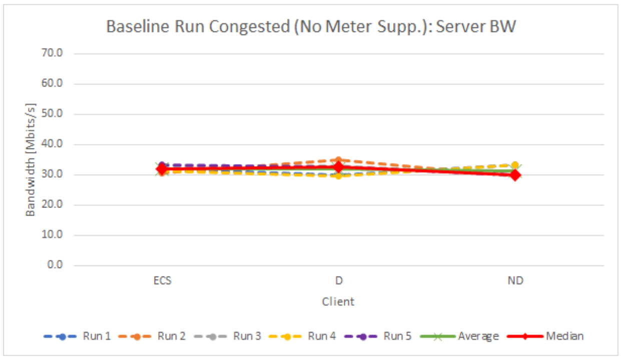
(c) Measured packet loss on receiver's end.

Figure B.2: Baseline congested scenario. Subfigures (a) and (b) show the measured bandwidth on each end user of the communication, and subfigure (c) shows the packet loss of each communication type. iPerf is configured to create UDP streams of 100 Mbits/s on each sender. The experiment was performed five times and the graphs show the measured values of each run, the average value and the median.

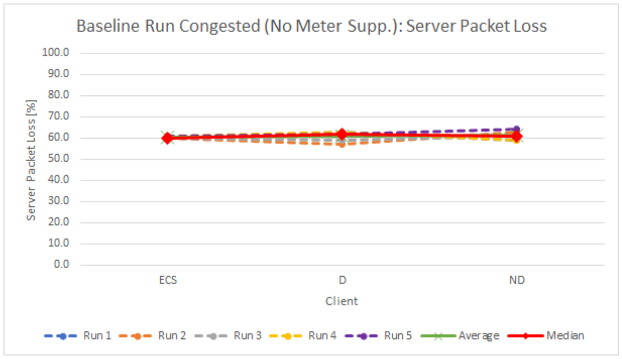
B.3 Baseline congested scenario with no meter support



(a) Measured bandwidth on sender's end.



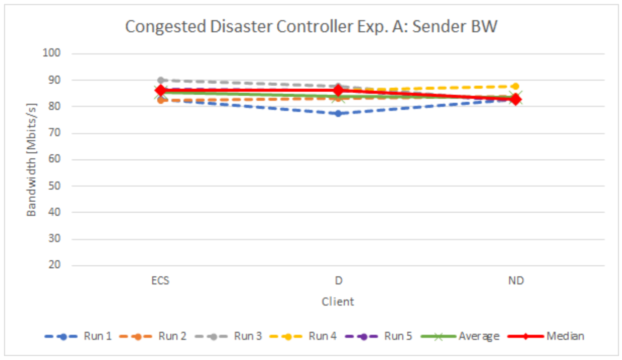
(b) Measured bandwidth on receiver's end.



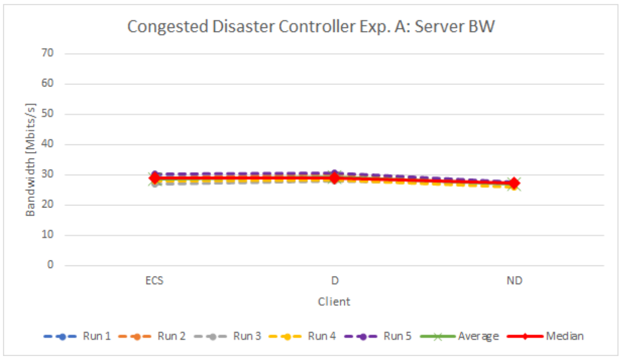
(c) Measured packet loss on receiver's end.

Figure B.3: Baseline congested scenario, without switch configuration to support meters. Subfigures (a) and (b) show the measured bandwidth on each end user of the communication, and subfigure (c) shows the packet loss of each communication type. iPerf is configured to create UDP streams of 100 Mbits/s on each sender. The experiment was performed five times and the graphs show the measured values of each run, the average value and the median.

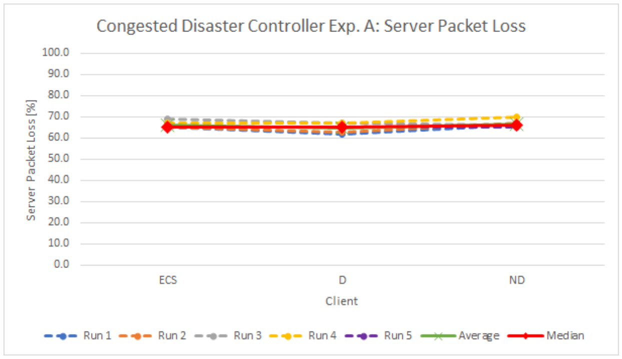
B.4 Disaster resilient controller: Experiment A



(a) Measured bandwidth on sender's end.



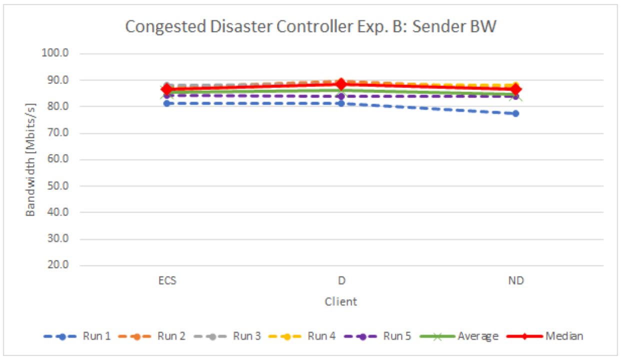
(b) Measured bandwidth on receiver's end.



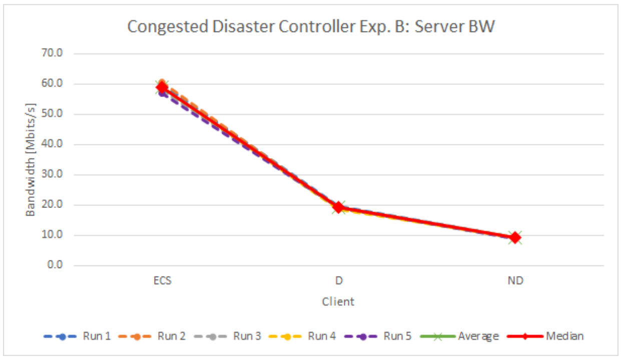
(c) Measured packet loss on receiver's end.

Figure B.4: Designed disaster controller in congested scenario. Subfigures (a) and (b) show the measured bandwidth on each end user of the communication, and subfigure (c) shows the packet loss of each communication type. iPerf is configured to create UDP streams of 100 Mbits/s on each sender. The controller limits the bandwidth of ECS, disaster and non-disaster commnunication by 33 Mbits/s each. The experiment was performed five times and the graphs show the measured values of each run, the average value and the median.

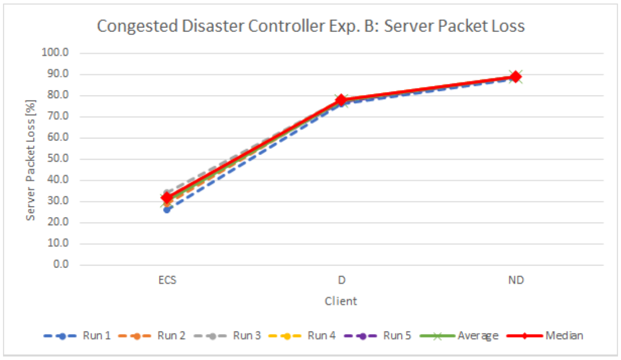
B.5 Disaster resilient controller: Experiment B



(a) Measured bandwidth on sender's end.



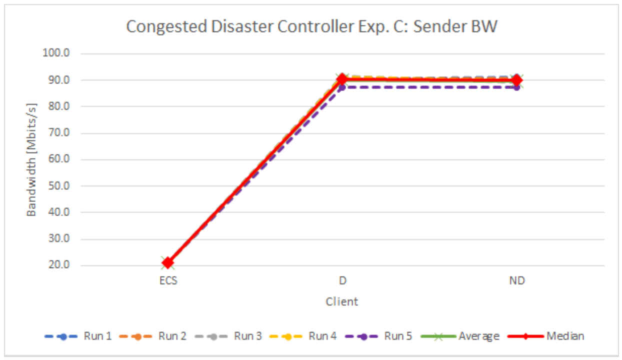
(b) Measured bandwidth on receiver's end.



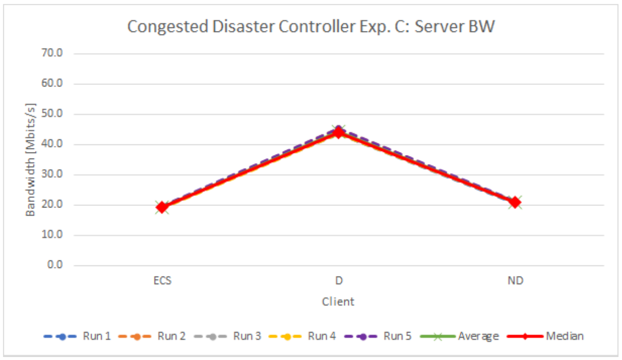
(c) Measured packet loss on receiver's end.

Figure B.5: Designed disaster controller in congested scenario. Subfigures (a) and (b) show the measured bandwidth on each end user of the communication, and subfigure (c) shows the packet loss of each communication type. iPerf is configured to create UDP streams of 100 Mbits/s on each sender. The controller limits the bandwidth of ECS, disaster and non-disaster commnunication by 70 Mbits/s, 20 Mbits/s and 10 Mbits/s respectively. The experiment was performed five times and the graphs show the measured values of each run, the average value and the median.

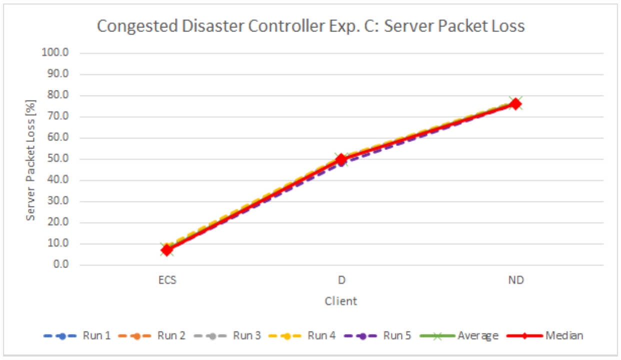
B.6 Disaster resilient controller: Experiment C



(a) Measured bandwidth on sender's end.



(b) Measured bandwidth on receiver's end.



(c) Measured packet loss on receiver's end.

Figure B.6: Designed disaster controller in congested scenario. Subfigures (a) and (b) show the measured bandwidth on each end user of the communication, and subfigure (c) shows the packet loss of each communication type. iPerf is configured to create UDP streams of 20 Mbits/s on ECS sender and 100 Mbits/s for disaster and non-disaster senders. The controller limits the bandwidth of ECS, disaster and non-disaster commnunication by 20 Mbits/s, 50 Mbits/s and 30 Mbits/s respectively. The experiment was performed five times and the graphs show the measured values of each run, the average value and the median.